

**Intuitionistic fuzzy XML query matching and rewriting**

**Mohammedsharaf Alzebdi**

School of Electronics and Computer Science

This is an electronic version of a PhD thesis awarded by the University of Westminster.

This is an exact reproduction of the paper copy held by the University of Westminster library.

---

The WestminsterResearch online digital archive at the University of Westminster aims to make the research output of the University available to a wider audience. Copyright and Moral Rights remain with the authors and/or copyright owners.

Users are permitted to download and/or print one copy for non-commercial private study or research. Further distribution and any use of material from within this archive for profit-making enterprises or for commercial gain is strictly forbidden.

---

Whilst further distribution of specific materials from within this archive is forbidden, you may freely distribute the URL of WestminsterResearch:  
(<http://westminsterresearch.wmin.ac.uk/>).

In case of abuse or copyright appearing without permission e-mail  
[repository@westminster.ac.uk](mailto:repository@westminster.ac.uk)

# **INTUITIONISTIC FUZZY XML QUERY MATCHING AND REWRITING**

**MOHAMMEDSHARAF ALZEBDI**

A thesis submitted in partial fulfilment of the  
requirements of the University of Westminster for the  
degree of Doctor of Philosophy

**December 2013**

## **Dedication**

I dedicate this thesis to my occupied country, Palestine. If I do not live in you, you have always lived in me. One day you will be free.

I secondly dedicate it to my family, particularly to my mum and to the soul of my father. Without you I would have not been what I am.

## **Acknowledgment**

I would like to express my gratitude and gratefulness to my director of studies Dr. Panos Chountas. I cannot find words to thank you with. You were the best supervisor I could ever have. Thanks for all the good times and memories. Also, special thanks go to Dr Andrzej Tarczynski for his useful instructions and advices; and to Fardous Bahbough for proofreading this thesis.

Very special thanks to the Harath-Taha group, Ihsan Abu-Showeb, Raed Amro, Mustafa HajMohammad and Mahmoud Alzabadi and to the Arab-ISH group and to all my friends and well-wishers in London and all over the world.

Finally, a big thank you to the University of Westminster for sponsoring my PhD.

## **Declaration**

The work included in this thesis is the author's own. It has not been submitted in support of an application for another degree or qualification of his or any other university or other institution of learning.

Signed:

Date:

## Abstract

With the emergence of XML as a standard for data representation, particularly on the web, the need for intelligent query languages that can operate on XML documents with structural heterogeneity has recently gained a lot of popularity. Traditional Information Retrieval and Database approaches have limitations when dealing with such scenarios. Therefore, fuzzy (flexible) approaches have become the predominant.

In this thesis, we propose a new approach for approximate XML query matching and rewriting which aims at achieving soft matching of XML queries with XML data sources following different schemas.

Unlike traditional querying approaches, which require exact matching, the proposed approach makes use of Intuitionistic Fuzzy Trees to achieve approximate (soft) query matching. Through this new approach, not only the exact answer of a query, but also approximate answers are retrieved. Furthermore, partial results can be obtained from multiple data sources and merged together to produce a single answer to a query. The proposed approach introduced a new tree similarity measure that considers the minimum and maximum degrees of similarity/inclusion of trees that are based on arc matching. New techniques for soft node and arc matching were presented for matching queries against data sources with highly varied structures.

A prototype was developed to test the proposed ideas and it proved the ability to achieve approximate matching for pattern queries with a number of XML schemas and rewrite the original query so that it obtain results from the underlying data sources. This has been achieved through several novel algorithms which were tested and proved efficiency and low CPU/Memory cost even for big number of data sources.

## Table of Contents

<b>Dedication .....</b>	<b>ii</b>
<b>Acknowledgement .....</b>	<b>iii</b>
<b>Declaration .....</b>	<b>iv</b>
<b>Abstract .....</b>	<b>v</b>
<b>Table of Contents .....</b>	<b>vi</b>
<b>List of Figures .....</b>	<b>ix</b>
<b>List of Tables .....</b>	<b>xii</b>
<b>1. Introduction .....</b>	<b>2</b>
1.1. Research objectives .....	2
1.2. XML Schema Heterogeneity .....	4
1.3. IF Approximate XML Query Matching .....	9
1.4. Thesis Structure .....	10
<b>2. XML Data Model and XML Queries .....</b>	<b>13</b>
2.1. Introducing XML .....	13
2.2. XML Data Model .....	15
2.3. Flexibility of XML Models .....	19
2.4. XML Query Languages .....	21
2.5. Chapter Summary .....	24
<b>3. XML Query Matching .....</b>	<b>26</b>
3.1. Overview .....	26
3.2. Traditional Schema Matching .....	27

3.3. XML Schema Matching/Similarity .....	28
3.4. XML Query Matching Approaches .....	31
3.4.1. Tree Edit Distance .....	32
3.4.2. Pattern Tree Matching .....	34
3.4.2.1. Twig pattern matching for query processing .....	35
3.4.2.2. Structural Pattern Tree Matching .....	40
3.4.2.3. Extended Pattern tree matching .....	43
3.4.3. Graph Pattern Matching (GPM) .....	46
3.4.4. XML Query Relaxation .....	48
3.4.5. Tree Algebra for XML (TAX) .....	52
3.4.6. Pattern Tree Mining .....	54
3.5. XML Query Rewriting .....	56
3.6. Chapter Conclusion .....	59
 <b>4. Intuitionistic Fuzzy Trees .....</b>	 <b>62</b>
4.1. Intuitionistic Fuzzy Logic (IFL) .....	62
4.2. Intuitionistic Fuzzy Trees (IFT) .....	64
 <b>5. Intuitionistic Fuzzy Pattern Tree Matching .....</b>	 <b>74</b>
5.1. Overview .....	74
5.2. Soft Node Matching .....	77
5.3. Soft Arc Matching .....	81
5.3.1. Direct Match .....	83
5.3.2. Inverted Match .....	83
5.3.3. AttNode Match .....	84
5.3.4. Normalized Match .....	85
5.3.5. Separating Node Match .....	87
5.3.6. Hybrid arc Match .....	89



5.4. Pattern Tree Matching Matrices .....	96
5.4.1. Node Mapping Matrix (NMM) .....	96
5.4.2. Arc Mapping Matrix (AMM) .....	98
5.4.3. Query Index Matrix (QIM) .....	99
5.5. Chapter Conclusion .....	104
<b>6. XML Query Rewriting .....</b>	<b>106</b>
6.1. Composing Queries .....	106
6.2. Query Rewriting Algorithms .....	109
6.3. Query Ranking .....	115
6.4. Chapter Conclusion .....	116
<b>7. Experimental Results .....</b>	<b>118</b>
7.1. Prototype .....	118
7.2. Mapping Phase .....	119
7.3. Query Filtration and Ranking .....	123
7.4. Query Rewriting Phase .....	126
7.5. Processing Cost .....	128
7.6. IFT vs. Other approaches .....	133
7.7. Chapter Conclusion .....	135
<b>8. Conclusion and Further Work .....</b>	<b>137</b>
8.1. Summary .....	137
8.2. Contributions and Limitations .....	139
8.3. Directions for Further Work .....	141
<b>References .....</b>	<b>143</b>
<b>Appendix A: Publications .....</b>	<b>151</b>

## List of Figures

Figure 1: A sample of books in an online shop XML database .....	5
Figure 2: An XQuery to return book details .....	5
Figure 3: Matching Pt with several data schemas .....	7
Figure 4: Departments' info. XML as (a) document, (b) tree .....	16
Figure 5: Departments' info. DTD as (a) document, (b) tree .....	18
Figure 6: Modelling a node as (a) element or (b) attribute, (c) Treating data as meta-data .....	19
Figure 7: Different ways of connecting XML elements .....	20
Figure 8: Normalised vs. non-normalised XML schemas .....	21
Figure 9: An XPATH query example .....	22
Figure 10: TED between two trees .....	32
Figure 11: An XML document indexed with T-index and E-index ...	36
Figure 12: (a) Original twig T, (b) Predicate Part $T_{pred}$ (c) Output part $T_{out}$ .....	39
Figure 13: Twig queries with AND/OR predicates .....	44
Figure 14: Twig queries with (a) wildcard, (b) Not predicate and (c) Following-sibling predicate .....	45
Figure 15: A graph model of an XML document with ID/IDREF ...	45
Figure 16: Query relaxation techniques .....	49
Figure 17: (a) collection C, (b) pattern P, (c) result of $\sigma_{P, SL}(C)$ , (d) result of $\pi_{P, PL}(C)$ .....	54
Figure 18: Fuzzy frequent subtrees .....	56
Figure19: (a) Normal tree vs (b) IFT .....	66

Figure 20: Tree inclusion of $T_1$ into $T_2$ .....	69
Figure 21: IFT Inclusion Algorithm .....	72
Figure 22: A pattern tree with two different schema trees .....	76
Figure 23: Soft Node Matching Algorithm .....	79
Figure 24: Types of soft arc matching .....	81
Figure 25: Normalised arc match .....	86
Figure 26(a) A pattern $P_t$ , (b) a schema tree $S_t$ .....	88
Figure 27: AttNode-SepNode hybrid match .....	90
Figure 28: Different combinations of Hybrid arc match .....	91
Figure 29: Soft Arc Matching Algorithm – part 1 .....	94
Figure 29: Soft Arc Matching Algorithm – part 2 .....	95
Figure 30: Node Mapping .....	97
Figure 31: The NMM for mappings in figure 30 .....	97
Figure 32: Arc Mapping .....	98
Figure 33: AMM for mappings in figure 18 .....	99
Figure 34: (a) QIM, (b) AMM .....	101
Figure 35: Generate New Queries Algorithm .....	102
Figure 36: Different matching twigs .....	103
Figure 37: Sibling, parent, child, ancestor and descendant arcs .....	107
Figure 38: Joining matching twigs .....	108
Figure 39: An XQuery example .....	110
Figure 40: Query Rewriting Algorithm .....	113
Figure 41: $P_t$ 's with different sizes .....	119

Figure 42: Remaining queries in QIM .....	125
Figure 43: A new query with components of FLWOR expression ...	127
Figure 44: Performance results .....	128
Figure 45: Memory consumption results .....	129
Figure 46: Heap memory usage during execution time for 10 DTDs	130
Figure 47: Heap memory usage during execution time for 50 DTDs	131

## List of Tables

Table (1): Results of node and arc mapping .....	120
Table (2): IFT vs. Other approaches .....	134

# **Chapter One: Introduction**

## **1. Introduction**

- 1.1. Research objectives
- 1.2. XML Schema Heterogeneity
- 1.3. IF Approximate XML Query Matching
- 1.4. Thesis Structure

# **1. Introduction**

The internet is undoubtedly the biggest data source ever with huge amounts of data from different sources following different formats. One of the main challenges in computer science is how to make data sharing and exchange between these sources possible; or in other words, how to develop a system that can deal with all these differences in data representation and extract useful knowledge from there. And since XML is the de facto standard for representing data on the internet, XML query matching has gained so much popularity recently [15- 26].

In this thesis, we propose a new approach for approximate XML query matching that aims at achieving soft matching of XML queries with XML data sources; thus overcoming the issue of querying heterogeneous XML documents.

## **1.1.Research objectives:**

The thesis aims at presenting a novel approach for approximate XML query matching that can resolve high structural diversity in XML data sources. Particularly, the research objectives of this thesis are the following:

- To propose a new graph-based approach for approximate matching of XML queries based on matching nodes and arcs of

a pattern tree i.e. an XML query, with a set of XML data schemas (DTDs).

- To be able to obtain partial results from multiple data sources and join them together in order to construct an answer to a pattern query.
- To redefine *support* and *confidence* to reflect the amount of matching nodes and arcs respectively, resulting with a two-value measure that indicates to the maximum degree of matching (Fuzzy Support) and the minimum degree of matching (Fuzzy confidence).
- To provide new techniques that softly match arcs, as basic structural components of XML schemas, without the need of two arcs being exactly matching, and then combine these arc together to construct answers to the original query.
- To develop a novel algorithm for rewriting a pattern query into new ones in the light of node and arc matchings.
- To develop a novel algorithm to rank the new queries depending on their precision (confidence) or performance according to users' requirements.



## **1.2.XML Schema Heterogeneity**

XML (eXtensible Markup Language) is W3C Recommendation considered as the standard format for structured documents and data on the Web. It is extensible because it is not a fixed format like HTML, which makes it possible to define new tags. Unlike HTML, XML documents consist of data and description of that data (Meta data) in a text format. While HTML was designed to display data, XML was mainly developed to structure, transport and store data [1]. Given that XML is the most common standard for data transmissions between heterogeneous systems, it has gained great popularity recently, especially in web applications.

As the amounts of data transmitted and stored in XML are rapidly growing, the ability to efficiently query XML is becoming increasingly important. Several XML query languages have been proposed for that purpose such as XML-QL, YATL, Quilt, Lorel and XQuery[2]. Those have provided good results; however, there are still some performance-related and structural heterogeneity challenges that need to be addressed before these languages can be mature enough.

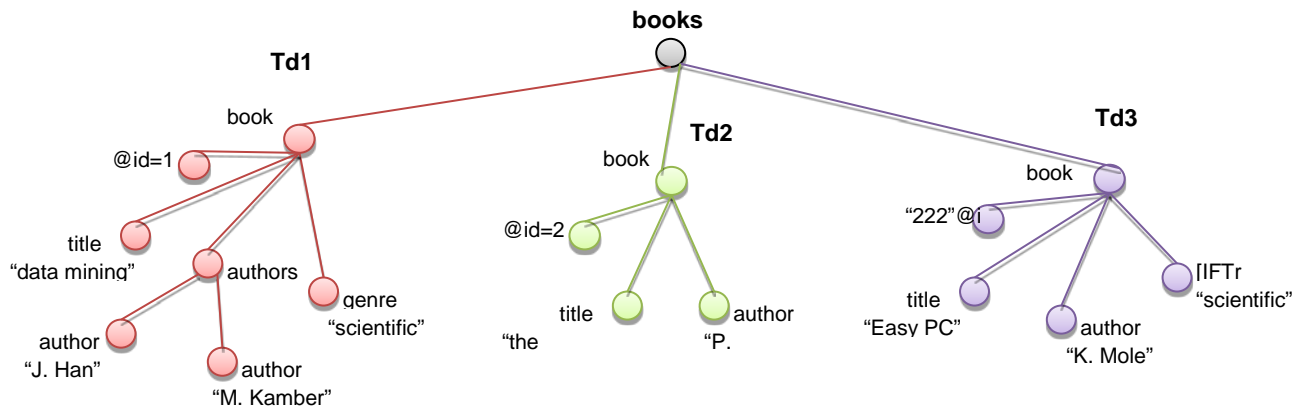


Figure 1: A sample of books in an online shop XML database

Figure 1 shows the details (schemas) of 3 books each belongs to a different data source. In order to retrieve data from those sources using XQuery, a general query is formed according to the user's understanding of the domain, without being aware of the underlying schemas. For example, a query that returns the book titles, authors and categories will look like this:

```
for $bin (doc('books.xml')//book)
return
<book>
  <title>{data($b/title)}</title><author>{data
($b/author)}</author>
<category>{data($b/genre)}</category>
</book>
```

Figure 2: An XQuery to return book details

Unfortunately, the query will only return details of books 2 and 3 but not book 1 because books 2 and 3 have matching structures with the query,

whereas book 1 does not. Looking at the schema of book 1, there is no *author* child for that book. Additionally, the book *category* is labelled as “*genre*” and the XQuery engine cannot recognise that it is a synonym for “*category*”.

The above example is one form of heterogeneity in XML schemas, however, many other forms can be identified (see page 8), and those are in need of XML Query languages to address them. Suppose that we are interested in finding information about university departments with research groups along with any projects and/or publications of these groups. According to our understanding of that domain, and without knowing the structure of underlying data sources, we might form a query that looks like Pt in Figure 3 below. Nodes with single circle shape indicate structural nodes that are not part of the output, whereas double-circled ones refer to output nodes. Node labels that are underlined, e.g. dname and @id, signify ID nodes acting as Primary Keys.

In some cases, it might not be possible to find an answer to your query based on one data source. In our motivating example shown in Figure 3, an answer to Pt needs to be obtained from three different sources s1, s2 and s3 with schemas (DTDs). These represent information about departments, projects and publications respectively. The challenge now is how to match Pt to

different parts of the data sources, and how to rewrite the query so that it retrieves data from these sources.

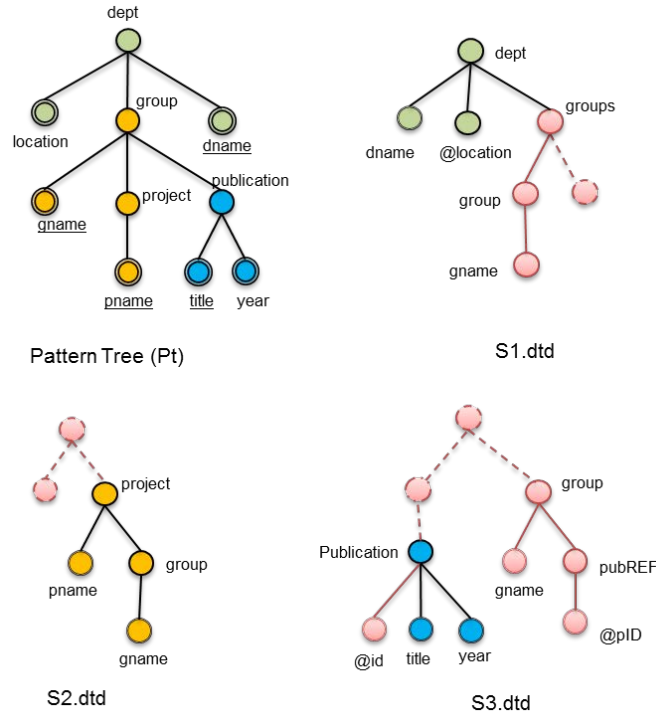


Figure 3: Matching Pt with several data schemas

Looking at how subtrees (twigs) of Pt are matched to the data sources (schemas) s1.DTD, s2.DTD and s3.DTD, we can see that twig 1, department's information, can be matched to s1. However, the element node *location* in Pt needs to be matched to the attribute node *@location* in S1. For twig 2, it can be noticed that the arc (group, project) in Pt is structured as (project, group) in s2. Lastly, twig 3 can be fully and directly matched to the correspondent twig in s3; however, we cannot determine which publication belongs to which group because the arc (group, publication)

does not have a match. Nevertheless, there is an indirect connection between the group and publication using the ID/IDREF directives.

To sum up, the forms of heterogeneity in XML data schemas can be:

- Representation of a certain domain can be scattered in multiple schemas instead of one single schema.
- A node, such as '*location*', can be modelled either as an element node or attribute node, and this is mainly due to flexibility of XML. However, if the node is planned to have child(ren) then it has to be an element node.
- Many-to-many relationships between two nodes, such as *group* and *project*, can be modelled as an arc (group, project) or (project, group). Even in case of one-to-many relationships, two nodes can still be modelled differently.
- Sometimes separating node(s) can be found between a parent and a child node e.g. the arc (dept, group) in Pt can be matched with the arc (dept, group) in S1 even though there is a separating node (*groups*) between the parent and the child.
- Some XML documents are *normalised* i.e. ID/IDREF are used to connect “entities” together, just like primary and foreign key connections in relational databases.

The above forms of heterogeneity in XML schemas can often be found in reality, especially when schemas belong to different sources. Everyone has his own perception of a certain domain, and s/he models it in a different way. Even though the literature is full of studies presenting approaches to handle heterogeneity in XML schemas, some of the above forms of diversity have not been addressed yet, to the best of our knowledge.

### **1.3. Intuitionistic Fuzzy XML Query Matching**

Because of heterogeneity in XML schemas, traditional crisp querying techniques are not efficient for analysing XML data, because they require exact query matching. Therefore, there is a need for new approaches that can achieve approximate query matching instead. Through these new approaches, not only the exact answer of a query, but also approximate answers will be retrieved.

Even though many studies have addressed approximate query matching in the literature [15-20, 23, 26, 30]; we believe that their approaches have some limitations, while an Intuitionistic Fuzzy approach is very useful to achieve approximate XML query matching by considering matching a pattern tree with multiple data sources and then joining sub-results together in order to construct a complete answer to a query. The focus of this thesis is on matching schemas rather than contents of XML documents, based on

Soft Node Matching as well as Soft Arc Matching. The degree of query matching is specified by redefining two measures, support and confidence. Matching Pt is mainly based on the primitive tree structure, arc, meaning that an answer of a query can be constructed from different arcs or twigs, probably from different sources, by joining these twigs together. New methods of matching arcs are presented in this research along with new algorithms to rewrite the original query so that it can return data from multiple data sources based on the matching output.

#### **1.4. Thesis Structure**

This thesis is structured as follows. The first chapter presents an overview of the problem domain, research objectives and the proposed solution. In chapter 2, the XML model and XML Queries are addressed along with a review of XML's main features. A comprehensive literature review is demonstrated in chapter 3 covering traditional schema matching with main focus on XML similarity and XML pattern tree matching approaches, particularly structural matching approaches. Additionally, relevant query rewriting approaches are presented and discussed thoroughly. In chapter 4, IFT is introduced together with a set of formal definitions to illustrate a novel approach of approximate similarity matching between two trees. The main contribution of this thesis is in chapter 5 where a novel approach for soft node and arc matching is introduced. Formal definitions were presented

for different type of soft arc matching. Furthermore, node and arc mapping matrices are introduced. In chapter 6, novel algorithms are developed for efficient rewriting of the original query based on the output of arc matching. The proposed approach is implemented and tested in chapter 7 and results are demonstrated. Finally, in chapter 8, a summary of the thesis is shown accompanied by the main contributions, limitations and further research directions.



## **Chapter Two: XML Data Model and XML Queries**

### **2. XML Data Model and XML Queries**

#### **2.1.Introducing XML**

#### **2.2.XML Data Model**

#### **2.3.Flexibility of XML models**

#### **2.4.XML Query Languages**

#### **2.5.Chapter Summary**

## **2. XML Data Model and XML Queries**

In this chapter, we present the XML model and XML queries along with the main features of XML. Section 2.1 introduces an overview of XML and its applicability whereas section 2.2 addresses the XML data model and XML schemas. Section 2.3 reviews the flexibility in XML and points out its main benefits and pitfalls. Finally, in section 2.4 XML Query languages are discussed.

### **2.1.Introducing XML**

When XML was first invented in 1998, its main purpose was mostly to be a format for web pages and other narrative documents intended to be read by people [3]. The main advantage was that data was stored separately from web page templates, allowing development of web pages on the fly by storing data in changeable XML documents that can be updated at any time without updating the actual HTML web page design.

Not long after, XML became of more significance, much more than just being storage for changeable web data. First and most of all, XML has become the solution of the biggest challenge in data sharing and integration, platform incompatibility. Because it has both data and semantics of the data (meta-data), XML has made data more portable and allowed different

software applications and systems to exchange data easily. Before XML, the typical solution was writing a custom code to transfer the data from one system to another, which was inefficient.

Furthermore, many applications on the internet, as well as on local computers, use XML documents to manage certain processes. For example, XML files are used to perform installation and maintenance tasks for Microsoft Office 2010 [4]. For internet applications, web services operate heavily on XML content to communicate with other different applications[5].

It is worthwhile mentioning that XML is a semi-structured language, meaning that it is neither structured nor unstructured, it is somehow structured. Consequently, XML documents can be classified into two types according to the degree of structure: Document-centric and data-centric[6, 7]. The former, is less structured and it is a rich-text document; therefore, it is not developed to exchange, store or analyse data. It is mainly there for human consumption, not to be read by computers. Examples can be found in publications, reports, and web-pages with textual data. Data-centric documents on the other hand, are more structured and they use XML to represent data that is stored or transported between systems. Because they have good level of structure, data-centric documents are intended to be understood by computers.

Overall, XML has gained a big amount of popularity, mainly because of the following reasons:

- Simplicity of its syntax: this made it easy to learn and use.
- Flexibility: it allows developers to choose their own tags (semantics) and plan data schemas according to their needs.
- Complex structures can be represented easily, including hierarchical structures.
- Easy to develop and debug: since it is text-based, an XML document can be opened and edited using any basic text editor.
- Language and platform independent: It is now supported by most of the platforms including internet browsers, database systems and even mobile phones. These consist of tools to read, write and manipulate XML.

## **2.2. XML Data Model**

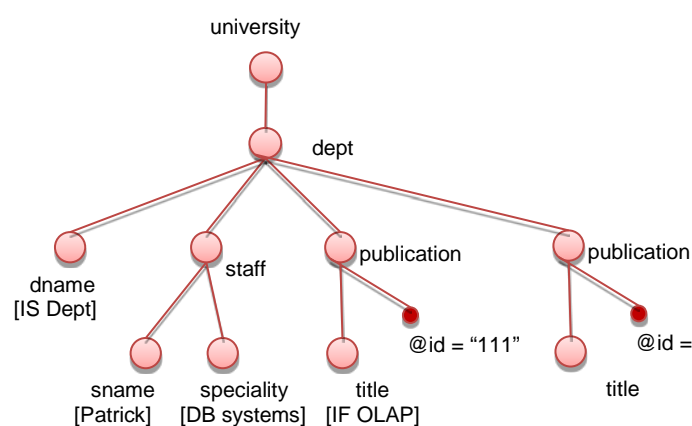
XML documents have a *tree-like* structure; however, in computer science literature, it is mostly referred to as simply *tree* [8]. This is due to the hierarchical structure of XML documents where each element (parent) can be composed of a number of elements (children) and each element can have no more than one parent. Some authors, however, argue that XML documents should be treated as graphs, rather than trees[9-11]. The reason

behind that stems from the different interpretations of the ID/IDREF connections in XML documents. The graph model supporters consider those connections as edges (arcs) and treat them just as any other edge. Having said that, the majority of previous studies treat XML documents as trees.

In figure 4, a snippet of XML document holding information about departments, staff and publication is shown along with a tree representation.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<university> <dept>
  <lname> IS Dept</lname>
    <staff>
      <sname>Patrick</sname>
      <specialty> DB Systems</specialty>
    </staff>
    <publicationid="111">
      <title> IF OLAP </title>
    </publication>
    <publicationid="222">
      <title>IFT Matching </title>
    </publication>
  </dept> </university>
```

(a)



XML documents consist of data and data about the data (meta-data). The latter provides semantics as well as schematic information about the meanings and relationships between different parts of XML documents (elements). However, due to the flexibility of XML e.g. optional elements, two documents representing the same domain might have different structures (schemas). Therefore, XML schema definitions were presented to specify precisely which elements should appear, where in the document and what the elements' contents and attributes are. Using a parser, each XML document is compared (validated) against a schema document, and if any difference found, the document will be considered invalid.

There are two types of XML schema definitions, DTD (Document Type Definition) and XSD (XML Schema Definition). DTDs were introduced first and they are still in use. A DTD can be within the XML document (Internal), or as a separate document (External). Written in a formal syntax (not XML syntax), DTDs describe the general structure of XML document with less constraints than that of XSDs. Overall, the main differences between the two types can be summarised by the followings:[5, 6]

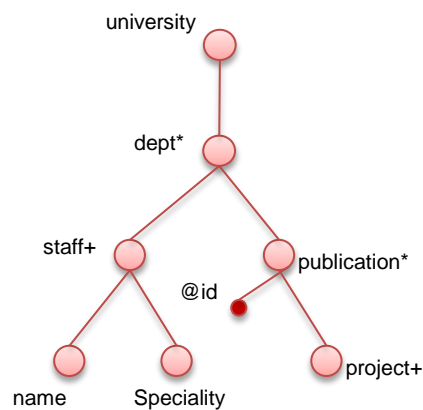
- XSDs use XML syntax where DTDs do not.
- In XSDs, elements hierarchies are explicitly specified unlike DTDs.

- XSDs have data typing capabilities whereas DTDs only use the text data type (PCDATA).
- XSDs can define precise cardinality constraints on elements whereas DTDs offer limited capabilities.

Figure 5 shows the DTD of the XML document in figure 4. Not only XML documents, but also DTDs can be modelled as trees.

```
<!ELEMENTuniversity(dept*)>
<!ELEMENTdept
  (dname,publication*,staff+)>
<!--ATTLISTpublicationid ID #REQUIRED-->
<!ELEMENTpublication (title)>
<!ELEMENTstaff      (name,speciality)>
<!ELEMENTdname      (#PCDATA)>
<!ELEMENTtitle      (#PCDATA)>
<!ELEMENTname       (#PCDATA)>
<!ELEMENTspeciality (#PCDATA)>
```

(a)



(b)

Figure 5: Departments' information DTD as (a) document, (b) tree

### 2.3. Flexibility of the XML model

One of the main reasons of XML popularity is the flexibility it offers for choosing tag and attribute names, cardinality and element nesting. Developers are allowed to choose their own tag and attribute names; they even have the freedom to model a data field as an attribute node or as an element node (Figure 3 (a) and (b)). Although allowing users to define their own tags sounds positive, it might result in users mixing between data and meta-data. For example, in an online auction website, the XML schema in figure 6 - (c) can be seen. The nodes Asia, Africa, S. America and N. America are modelled as meta-data (element nodes) even though they represent data referring to the continent where the auction had taken place.

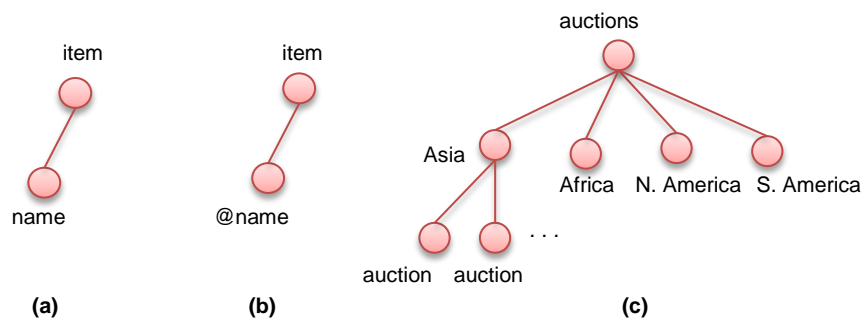


Figure 6: Modelling name as (a) element node or (b) attribute node, (c) Treating data as meta-data

Furthermore, the cardinality of each element can be specified, which allows for optional and multiple elements to be defined. This can cause two XML documents following the same schema to be highly different. For example, in



figure 7-(a), a *dept* element can have at least one or more *group* elements.

The followings are the options XML offers for children cardinality:

- ?: zero or one element is allowed
- \*: zero or more element is allowed
- +: one or more element is allowed
- If no suffix exists, then the cardinality of the element is one and only one.

Moreover, XML allows us to connect (nest) elements with no restrictions e.g. in figure 7 below, the *dept* element can be related to *group* element as either parent-child or child-parent. Whether the relationship between two elements is 1-1, 1-n or n-n, they still can be modelled differently according to the users perception, or point of interest.

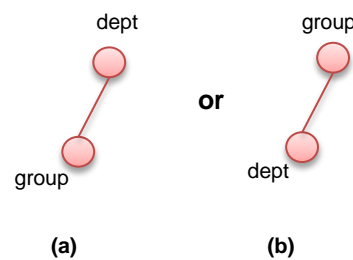


Figure 7: Different ways of connecting XML elements

Moreover, one might choose to have normalised or non-normalised XML documents. For small or medium size XML documents, it is acceptable to be un-normalised. For big size XML documents, in contrast, it is preferable to

use ID/IDREF connections in order to reduce redundancy and enable information integrity. Figure 8 shows a normalised versus an un-normalised XML schema representing the same domain.

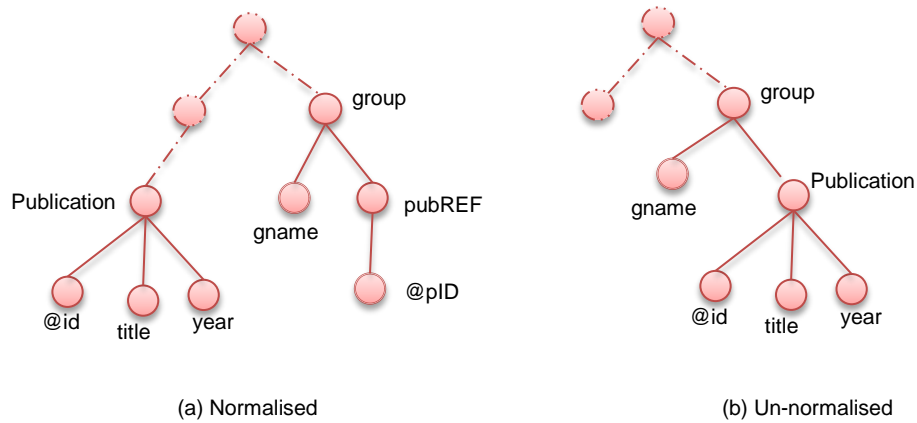


Figure 8: Normalised vs. non-normalised XML schemas

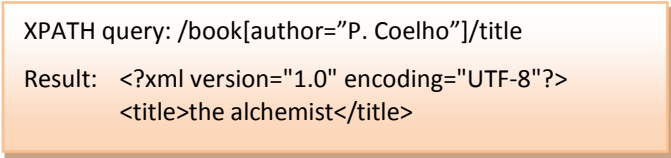
In essence, XML flexibility is a double-edged sword; it gives the option to model a certain domain according to user's perception, but it may result in schematic heterogeneity in XML documents as people tend to model the same information in different ways.

## 2.4. XML Query Languages

As the amounts of data transmitted and stored in XML are growing, the ability to efficiently query XML is becoming increasingly important. Several XML query languages have been proposed for that purpose such as XML-

QL, YATL, Quilt, Lorel, XPATH and XQuery[2]. Being recommendations of W3C, XPATH and XQuery are now the most predominant languages for XML queries.

XPATH is defined as a non-XML language for retrieving parts of XML documents [3]. In the XPATH data model, each document is represented as a tree of nodes, where there is one node called “root” and other nodes that have parent-child and ancestor-descendant relationships. XPATH expressions are used to navigate through this tree and retrieve nodes matching that expression in terms of structure and predicates. XPATH can support both simple and complex queries. For example, figure 9 below shows the result of applying an XPATH query on the books XML documents in figure 1.



```
XPATH query: /book[author="P. Coelho"]/title
Result: <?xml version="1.0" encoding="UTF-8"?>
       <title>the alchemist</title>
```

Figure 9: An XPATH query example

In 2007, XQuery, which is an extension of XPATH, was recommended by W3C making it the most popular language for querying XML data. According to [6], “XPATH2.0 and XQuery 1.0 support all of the same functions and operators, and they share the same data model”. XQuery is designed to allow the construction of concise, flexible and easily understood

queries that can operate on diverse XML data sources, including both databases and documents [2].

XQuery provides high capabilities to analyse data-centric XML documents, such as huge XML databases, offering the ability to filter, merge and order data. This can be beneficial for analysing “application logs, transaction logs and audit logs to identify potential application errors and security issues, and so on” [5]. Additionally, XQuery is an excellent solution for transforming data from internal application-specific formats to standard exchange format.

XQuery works by scanning through an XML document, applying predicates to the query and returning parts that match the query as a new XML document. Even though it is not yet finalised by W3C (World Wide Web Consortium) as a standard, XQuery is nowadays implemented in industry.

To query XML documents, a number of expressions can be used, the most powerful one is called FLWOR (for-let-where-order by-return) which is similar to the (select-from-where) clauses in relational SQL. Figure 2 in the previous chapter shows an example of an XQuery using FLWOR expression. Hence, FLWOR expressions consist of the following parts:

- For: specifies a list of XML nodes to iterate over, similar to the FROM clause of a SELECT statement
- Let: allows user to declare variables

- Where: contains expressions that perform filtering, similar to the WHERE clause in SQL.
- Order by: allows user to order results by a node(s) in ascending or descending order.
- Return: specifies the nodes that will be returned by the query.

Using FLWOR expressions, we can do much more than just retrieving elements from XML documents; we can join two parts (sub-trees) of an XML document or even of more than one document based on an equi-join. Furthermore, FLWOR expressions supports aggregate functions such as sum(), count() and all others supported by relational SQL.

## **2.5. Chapter Summary**

This chapter introduced the XML data model, XML features and XML query languages. Additionally, it presented a review of the significance of XML and its main features. The next chapter consists of literature review of related work.

## **Chapter three: XML Query Matching**

### **3. XML Query Matching**

#### 3.1.Overview

#### 3.2.Traditional Schema Matching

#### 3.3.XML Schema Matching/Similarity

#### 3.4.XML Query matching approaches

##### 3.4.1. Tree Edit Distance

##### 3.4.2. Pattern Tree Matching

###### 3.4.2.1. Twig pattern matching for query processing

###### 3.4.2.2. Structural Pattern Tree Matching

###### 3.4.2.3. Extended Pattern tree matching

##### 3.4.3. Graph Pattern Matching (GPM)

##### 3.4.4. XML Query Relaxation

##### 3.4.5. Tree Algebra for XML (TAX)

##### 3.4.6. Pattern Tree mining

#### 3.5.XML Query Rewriting

#### 3.6.Chapter conclusion

### 3. XML Query Matching

This chapter presents a comprehensive literature review of related work. It starts with traditional schema matching studies and then moves to XML schema and query matching. XML query matching and rewriting approaches are classified according to the purpose and the adopted technique. All relevant research is critically evaluated and pitfalls are highlighted.

#### 3.1. Overview

The issue of having different data structures (schemas) representing the same data is a very common problem in the world of information systems. Solving this problem has been one of the main subjects of research; especially that it has lot of applicability in several database application domains such as schema integration, data warehousing, E-commerce, semantic query processing, schema mediation and others[7]. The most common approach in the literature about previous works on addressing schema heterogeneity is *Schema Matching (Mapping)*. However, other approaches such as *Schema Mediation*[12, 13]and *Query Matching*[14-20]are also common. Since XML Queries have been modelled as trees (schemas), the literature of XML Query matching and Schema matching have a lot in common.

### 3.2. Traditional Schema Matching

Schema matching (mapping) is an operation that takes two schemas as input and produces a mapping between the elements of the two schemas that correspond semantically to each other[21]. Many schema matching approaches have been proposed and those have been classified into different categories based on the level of matching; whether it is instance-based (content)matching, schema-based(structure) matching or a combination of both[21, 22].Moreover, the schema-based matching can be classified into element-level and structure-level matching.

Element-level matching techniques deal with mapping individual elements (attributes) together based on certain criteria. In a survey done by authors of [22], the following techniques were identified:

- String-based: those depend on the element labels similarity for mapping
- Language-based: those use Natural Language Processing to analyse morphological properties of the input words
- Constraint-based: those consider constraints on elements such as data types, cardinality and keys to determine whether two elements are matching.



- Linguistic resources: such as lexical ontologies. Those provide semantics to element labels and compare how close these semantics are to each other (e.g. synonyms and hyponyms).

Structure-level techniques, on the other hand, deal with matching combinations of elements i.e. structures; and this approach is more common. To be able to obtain data from different sources, it is necessary to know the structure as well as the semantics used in each one. A typical solution was to develop a global schema, map it to local schemas and apply a set of transformations (translations) to translate a global query into a set of local queries, and finally merge the local results together and return it as one answer. A slightly different approach was to compare local schemas against each other and map elements of two schemas that are semantically equivalent to each other, which is common in P2P (Peer to Peer) applications where two systems exchange data in both ways, sending and receiving.

Schema matching approaches provided good results for relational database environments where local schemas are known. However, there are still a lot of challenges in the way of automating such solutions.

### **3.3. XML Schema Matching/Similarity**

There are, obviously, a lot of commonalities between traditional schema matching and XML schema matching approaches. However, XML has its

own speciality as a hierarchical data model having tree structure. Building on traditional approaches, some researchers tried to adapt relational schema matching approaches to work on XML models, other came out with new ones.

XML schema matching, also referred to as “XML similarity”, has been a hot topic of research in the last decade leveraged by the increasing role of XML as the best choice for data interoperability, especially on the web. Finding the similarity between two XML documents can be of great applicability in many domains such as version control and change management, data integration, document clustering, and IR (Information Retrieval) [23].

**Version control and change management** is one of the main areas where XML document similarity can be beneficial. It provides a means to detect change between different versions of a document, and represent this change as an XML document instead of having a new modified copy of the same document. This enables users to view any version of the original document at any time by simply applying some edit scripts (Deltas). Additionally, one can monitor an XML document and by using query subscription and notification systems, s/he can be notified about any change (e.g. a new item has been added to a catalogue). Moreover, deltas can be used for archiving purposes by simply storing a sequence of deltas along with the correspondent XML document. Furthermore, XML similarity can be used in mirroring

applications to reduce network traffic by computing and propagating only the different between the document version at the server and that at a mirror site.

Another application for XML similarity is **classification and clustering**. XML classification refers to relating XML documents collected from the internet to a set of XML schemas (such as DTDs) in an XML database. This can be useful in the case of having a number of XML databases exchanging XML documents among each other. A new XML document is compared against schemas within a database using XML similarity algorithms, and the document is assigned to the schema that best matches it. XML clustering, on the other hand, is a process that groups similar XML documents together which can improve data storage indexing [23], and thus speed up data retrieval. Clustering can also be utilized in XML schema extraction by allowing the construction of more accurate and specific XML schemas in each cluster.

**Data integration** is one of the most beneficial areas of XML similarity approaches. Even though XML is popular for sharing and exchanging data between heterogeneous data sources, two XML documents representing the same information can be structured differently. Therefore, it is curtail for data integration to compare XML documents and determine the similarity and difference between each pair.

Lastly, XML similarity is very popular in **IR systems** where XML queries are modelled as schemas/structures called pattern trees. To retrieve data from a certain XML document, a query (pattern tree) is compared against the underlying XML schema, and only if it is similar (matching), a result is returned. Next section (3.4.) addresses XML Query matching approaches thoroughly.

### **3.4. XML Query matching approaches**

Since XML Queries are modelled as Query Patterns or Pattern Trees (Pt) in computer science literature, XML Query matching research cannot be separated from XML schema matching. However, it is worthwhile to point out that a Pt is usually compared to part of an XML document which is sometimes referred to as Tree Inclusion instead of Tree Matching because a Pt is expected to be included within a Schema tree (St), without the need to be fully matching/similar to it. In this section we present the most common approaches of XML Query/Schema matching in both database and IR communities. There are plenty of surveys on previous approaches [23-26] where they were classified into different groups according to different criteria. In this section we classify previous works according to the technique used to achieve query matching.

### 3.4.1. Tree Edit Distance

Tree Edit Distance (TED) is one of the earliest approaches of schema matching/similarity. In addition to its applicability in database and IR systems, it is also used in computational biology, image analysis, automatic theorem proving, and compiler optimization[27]. TED measures the distance (or similarity) between two labelled trees T1 and T2 by calculating the cost of transforming T1 into T2. The cost is determined by a sequence of edit operations (S) required to turn T1 into T2. Those operations are performed on tree nodes and can be *relabel*, *delete* or *insert* a node. Figure 10 below shows the TED between two trees T1 and T2. The node *document* is relabelled to *book*, the node *category* is deleted and finally, the node *authors* is inserted into T1 so that it matches T2. Overall, three edit operations were needed, thus the TED between T1 and T2 is 3.

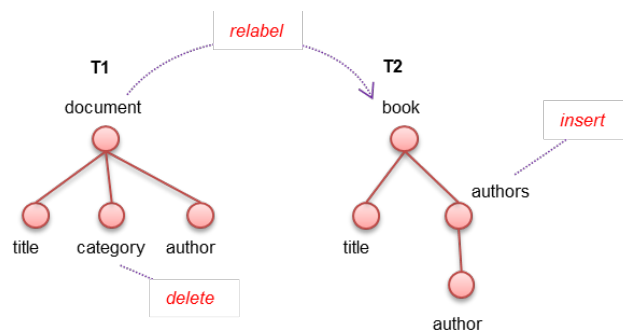


Figure 10: TED between two trees

Based on TED, Tree Inclusion is defined as follows[27]: T1 is said to be included in T2 if and only if T1 can be obtained from T2 by deleting some nodes from the latter.

Some previous studies used TED to compare XML documents and find similarity between them. In [28], authors used a TED based approach to compare pairs of XML documents within a collection and then cluster them according to the distance. In particular, their study addressed the case where two XML documents following the same DTD have different sizes due to optional and repeated XML elements. Using traditional TED approaches, such a pair of documents will have a big distance, and therefore will not be clustered together. The authors proposed a new approach that is based on edit operations not just on the node level, but also on the tree level. In addition to *relabel*, *insert* and *delete* node operations, *insert tree* and *delete tree* operations were proposed.

Chen and Chen [29] presented a new approach for tree inclusion that is based on deleting nodes from a target tree (T) in order to obtain a pattern tree (P). Deleting a non-leaf node can change T significantly by making two nodes parent-child instead of ancestor-descendant. Overall, authors claimed that their algorithm achieved significant improvement on performance as it needs less time than previous approaches without the need of extra space.

Where the aforementioned studies focused on XML structural similarity, some studies addressed the problem of XML content similarity. This is of great applicability in data cleansing, particularly in duplicate detection. In [30], authors proposed a study for resolving both XML instance and schema heterogeneity aiming at fuzzy duplicate detection. String Edit Distance similarity was utilized to calculate similarities between pairs of string values; whereas TED was used to address schematic similarity. To improve efficiency, three comparison (traversal) strategies were involved: i) Top-down comparisons: those are limited to XML elements having the same ancestors. ii) Bottom-up comparisons: because the XML data is usually stored in leafs, this might give a better performance. iii) Relationship-aware comparisons: those consider the elements influence on each other in both directions.

### **3.4.2. Pattern Tree Matching**

Many studies have been done on matching query patterns, some called it Tree Pattern (or Pattern Tree); others called it Twig Pattern. Therefore, the two terms will be used interchangeably hereafter. The proposed approaches vary significantly according to the intended purpose whether it is XML query optimisation i.e. cutting CPU and I/O cost[31-40], fuzzy query matching[41, 42] or structural query matching[43-50].

### 3.4.2.1. Twig pattern matching for query processing

Research on efficient pattern tree matching against XML data trees i.e. including structure and content matching focused on reducing CPU and I/O cost. Most of the previous work used indexing techniques to speed up query processing, particularly Inverted Index [39] with some enhancements. The classic inverted index maps a text word to a list, which enumerates documents containing the word and its position within each document. Zhang et al. [40] extended that by presenting a new algorithm, Multi-Predicate Merge Join (MPMGJN), which works on XML documents. Two types of indexes were introduced: T-index (from Text), which is similar to the aforementioned inverted index, and E-index (from Element) which maps XML elements to inverted lists that contain element names and positions within documents. Figure 11 shows an example of XML document indexed in that way. T-index has the format  $(docno, wordno, level)$ , whereas E-index has the format  $(docno, begin:end, level)$ , where *docno* is the document number; *wordno* is a number indicating to the word location; *begin* and *end* can be found by counting the start and end locations of a word (tag) in the document. For example, in figure 11, the node (tag) *dname* has an index value of (1, 3:5, 2) which indicates that *dname* is in document number 1, starting position is 3, ending position is 5 and in level 2.

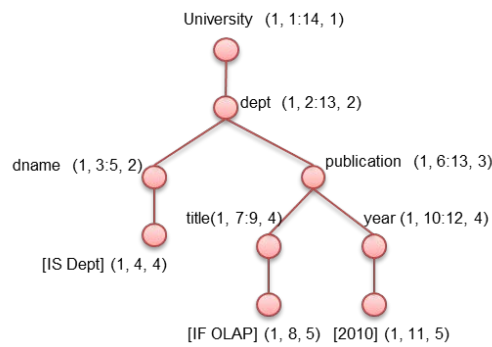


```

<university>
<dept>
  <lname> IS Dept</lname>
<publication>
  <title> IF OLAP </title>
  <year> 2010 </year>
</publication>
</dept>
</university>

```

(a)



(b)

Figure 11: An XML document indexed with T-index and E-index

Zhang et al. [40] paved the way to two main studies [31, 32], which then led the research in that area. In [31], authors proposed matching pattern tree queries by decomposing them into basic binary structural relationships, Parent-Child (PC) or Ancestor-Descendant(AD), match each part separately and then combine them together to construct an answer to the query. Two families of algorithms were proposed: tree-merge, and stack-tree join algorithms. Those algorithms focused on improving performance by

reducing the time required to find matchings between binary structures of pattern and data trees. Two input lists are extracted from  $P_t$ ,  $AList[a_1, a_2, \dots]$  (list of ancestors or parents) and  $DList[d_1, d_2, \dots]$  (list of descendants or children) are used by the algorithms above to generate an output list  $OutputList=[a_i, d_j]$  which consists of a list of pairs of ancestors and descendants. Overall, tree-merge algorithms have good performance sometimes but Stack-tree joins often provide optimal CPU and I/O performance. However, this approach has a disadvantage of producing big size of intermediate results even when the input and final result sizes are small.

The second main leading study [32] presented stack-based algorithms to achieve good performance and memory usage. A Pattern tree is decomposed into a number of paths (root-to-leaf) and those are processed using two main algorithms: PathStack and TwigStack. The former computes answers to a  $P_t$  by repeatedly constructing stack encodings of partial and total answers to the query path pattern. A stack  $S_q$  is created for each node  $q$  in  $P_t$ , and an answer is calculated by iterating through these stacks and matching the nodes from root to leaf. TwigStack, however, extends PathStack by including a phase for merge-joining the computed root-to-leaf paths in order to compute answers to the twig pattern. Authors claim that their approach does not produce a lot of intermediate results thus providing optimal performance.

More studies on efficient holistic twig joins built followed [32] and tried to improve it. Jiang et al. [33] states that the aforementioned approach can be improved by skipping elements that do not participate in the final twig match by using indexed XML documents and presented an approach to achieve that. More recently, Grimsmo et al. [34-36] proposed twig join algorithms that achieve worst-case optimality without affecting average performance. Particularly, the study addressed the effect of different filtering methods on performance and presented new data structures to improve filtering and thus yielding worst-case optimal performance.

A different approach of indexing XML documents was presented by Praveen in [37, 38] where XML documents and pattern twigs were transformed into sequences of labels by using Prufer's method which constructs a one-to-one correspondence between trees and sequences. Twigs are matched by simply matching the aforementioned sequences and finding all occurrences of a twig pattern sequence into an XML document sequence. Unlike some previous approaches, twigs are matched holistically i.e. without breaking them into root-to-leaf paths. Authors claim that the proposed approach results in correct answers as well as good performance.

While the above aforementioned studies focused on indexing techniques to improve query processing, a new approach based on using twig semantics was introduced by Bao et al. [51]. The approach utilises the schema

information (semantics) of XML documents as well as the twig query structure to speed up the query processing. Semantics of XML documents are captured from DTD/XSD and those can be *identifier constraints* (*unique* and *key* constraints) and *participation constraints* (*?*, *\**, *+*). These constraints are used to derive functional dependencies which are used to optimise query processing by stopping the query once the required match is achieved. The other optimisation technique works by breaking the query twig into two parts i) Predicate twig: This consists of nodes representing structural and content predicates that are not part of the output ii) Output twig: this consists of the nodes to be returned by the query. Figure 12 shows a twig query that is split into two parts.

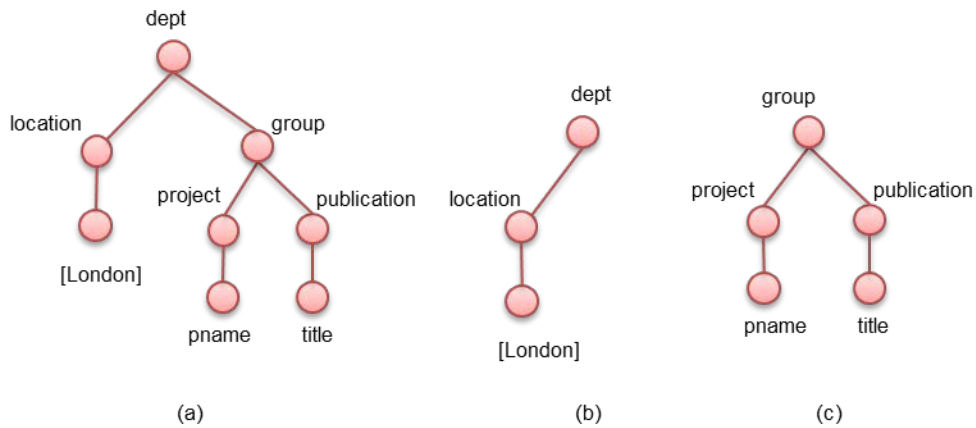


Figure 12: (a) Original twig T, (b) Predicate Part  $T_{pred}$  (c) Output part  $T_{out}$

In addition to approaches based on crisp XML databases, some authors presented new ideas for processing twig patterns on fuzzy XML data. Inspired by PathStack algorithm of Bruno et al.[32], Liu et al. [41, 42]

extended the indexing algorithm proposed in [40] to include fuzzy values to indicate the possibility of an XML element or a value. Authors point out two types of fuzziness in XML documents i) fuzziness in elements (structure): an XML element can be fuzzy if it is not a real element e.g. `<Val>` that does not exist in the non-fuzzy version of the document. In other words, it indicates to the degree of membership of an element in an XML document. ii) Fuzziness in attribute values (content): a certain value can have a fuzzy possibility e.g. `<age><Val poss=0.8> 27</Val></age>` means that the possibility of having age=27 is 80%.

Overall, the studies presented in this section considered different query matching approaches for the purpose of improving query performance rather than for matching pattern trees against source schemas. The next section presents approaches for schema-based pattern tree matching.

#### **3.4.2.2. Structural Pattern Tree Matching**

Some studies focused on matching the structure of a query pattern against the structures of a set of data sources in order to get over the structural heterogeneity of these sources. In this case, the pattern query is written based on one's common knowledge about the domain in hand without necessarily knowing the structure of the underlying sources. This is one of the biggest challenges for data interoperability between different systems.

While a great amount of research was dedicated to twig matching for query processing and optimisation purposes, less effort was focused on structural pattern matching. Tree Edit Distance based approaches were considered for calculating XML tree similarity and inclusion (See section 3.4.1); however, there were few attempts to calculate tree similarity by considering the number of common nodes and/or arcs. In this section we present few IR-style approaches on approximate structural matching of pattern trees.

Polyzotis[43] was one of the first researchers to address that problem. He proposed a study that focused on approximate answers for twig queries considering the structural part of the problem. His approach is based on a novel type of XML synopses called TreeSketches, which captures the key properties of the underlying path distribution and enables low error approximate answers. A new XML similarity metric, termed Element Simulation Distance (ESD), was proposed which, according to the author, outperforms previous syntax-based metrics by capturing regions of approximate similarity between XML trees. It considers both overall path structure and the distribution of document edges when computing the distance between two XML trees. According to that metric, two elements are considered to be more similar if they have more matching children, and less similar if they have fewer children in common. This can be reasonable when tree nodes have close semantics to each other. However, this is not always

the case. It is common to find nodes with weak semantic connections such as publication, group and department in figure 12-(a).

In other studies [44, 45], Sanz et al. proposed tree similarity measures between a pattern tree and sub-trees within an XML document as a solution for approximate XML query matching. Similarity is calculated based on the number of matching nodes without considering the semantics of parent-child connections. The process consists of two steps: first, identifying the portions of documents that are similar to the pattern (fragments and regions identification). Second, the structural similarity between each of these portions and the pattern is calculated. The proposed node similarity metric does not only depend on the label, but it also depends on the depth of the node “distance-based similarity”. Therefore, similarity linearly decreases as the number of levels of difference increase. However, when matching a pattern tree to an XML data tree, the hierarchical organization of the pattern and the region are not taken into account [45] i.e. matching is only on the node level but not on the arc/edge level.

Another significant study addressed element similarity metrics in structural pattern matching [46]. Authors introduced two types of element similarity measures, internal and external. The internal similarity measure depends of feature similarity which includes i)Node name similarity: this in turn can be classified into syntactic (label) and semantic (meaning) similarity. Node

names are first normalised into tokens that are compared using different string similarity approaches such as string edit distance. Semantic similarity, on the other hand, is calculated by using measures from WordNet ii) Data type similarity measure, iii) Constraint similarity measure, mainly cardinality constraint, and iv) Annotation similarity measure, which considers the provided annotations for tree nodes.

External similarity measure, on the other hand, considers the position (context) of the element in the schema tree i.e. it considers the element's relationship with the surrounding elements, descendants, ancestors, and siblings. A function was used to combine internal and external measures and give the overall similarity measure between two nodes.

Following a totally different approach, Agarwal et al. [47], proposes XFinder, a system for top K XML subtree search that works on exact and approximate pattern matching with focus on approximate structural matching between ordered XML trees. XML query trees as well as document trees were transformed into sequences which are then compared against each other. This technique was adopted in other studies as well[48-50].

#### **3.4.2.3. Extended Pattern Tree Matching**

Some studies proposed extending the expressiveness of pattern trees by not only including node labels and arcs, but also other features such as negation functions, wildcards and logical predicates. In [52], an approach for holistic



processing of twigs with AND/OR predicates was presented. Such twigs consisted of ‘AND’ and ‘OR’ nodes that can have multiple child nodes as shown in figure 13. Novel algorithms were developed to efficiently evaluate these twig queries on XML data that are either sorted or indexed for achieving high performance. Xu et al. [53] extended the previous study by proposing a twig query with AND/OR/NOT predicates, which they called XPattern. Authors proposed a path-partitioned indexing scheme to capture the path information of XML documents and used two relational tables for that purpose, one for encoding paths and another for encoding elements. A novel holistic algorithm called MPTwig was developed based on both path-partitioned encoding scheme and XPattern.

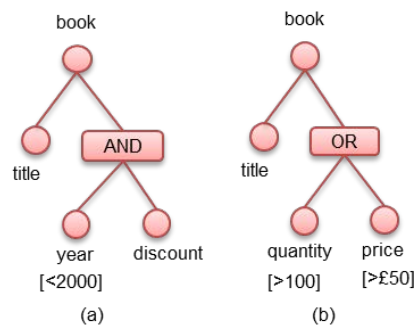


Figure 13: Twig queries with AND/OR predicates

In addition to twigs with logical operators, some studies considered further extensions. In [54], a framework for processing twigs with wildcards, negation functions and order restriction was introduced. Figure 14 shows three different extended twigs where in (a) a twig with wild card represents a

query for all nodes that have an author and title child nodes, (b) returns books with *title* child nodes but with no *discount* child nodes and (c) returns articles with a title child node and an *author* child node with order restriction. The main aim of the study is to achieve optimal XML pattern tree matching that outperforms TwigStack algorithm presented in [32]. Authors claim that TwigStack has a shortcoming in some cases where a choice has to be made between having possible intermediate results and missing potential correct answers, and that their algorithm overcomes this shortcoming. According to the authors, experimental results showed that their algorithms can correctly process extended XML twigs while keeping high performance and low size intermediate results.

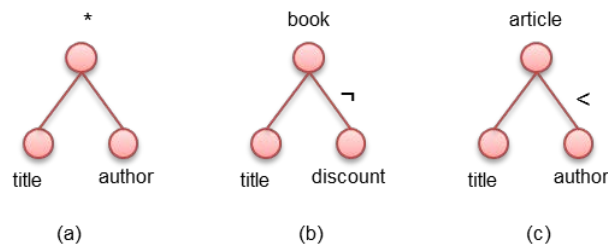


Figure 14: Twig queries with (a) wildcard, (b) Not predicate and (c) Following-sibling predicate

Zeng et al. [55] proposed an extended pattern approach called Generalised Tree Pattern Queries (GPTQs) to handle XML documents with ID/IDREF connections, which were treated as graphs. Pattern tree nodes were classified into backbone nodes (specifying nodes with no predicates) and predicate nodes (nodes with value predicates) and functions were defined to process each type. Authors introduced pruning algorithms to reduce the size of

intermediate results and the number of required join operations in addition to an indexing technique for finding reachability between graph nodes.

### **3.4.3. Graph Pattern Matching (GPM)**

Whereas most of previous works on XML query matching treated XML documents as tree structures, others argued that node-labelled graph structure, particularly Directed Acyclic Graphs (DAGS), is more appropriate. This was mainly due to ID/IDREF connections between XML elements, which makes it have a graph shape, as shown in figure 15. Therefore, some studies proposed extending Tree Pattern matching to Graph Pattern matching. In addition to its applicability in query matching, Graph Pattern matching can be useful in other areas such as keyword search in XML documents [56], finding patterns in web-services connection, relationships in social networks, research collaboration patterns, publication citation connections [10] and even in gene ontology research [57].

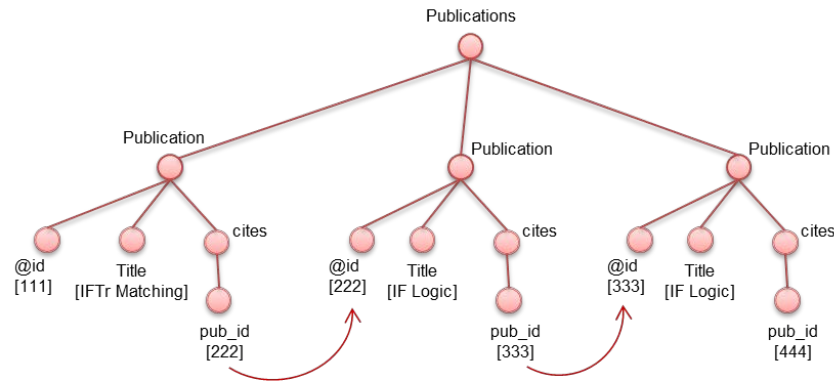


Figure 15: A graph model of an XML document with ID/IDREF

Researchers proposed different approaches for implementing GPM. Chen et al. [57] extended the twig join approach of [32] to work on DAGs and used that to present extensions to XPATH queries. Authors of [57] developed a set of stack-based algorithms to handle path, twig and DAG patterns to achieve exact query matching. In a different GPM study, Kimelfeld et al. [9] proposed a query language that incorporated filtering (excluding semantically weak matches) and ranking mechanisms while preserving the simplicity and efficiency of twig queries. Another two-step approach was presented in [10, 11] but it consisted of a filter step and a fetch step. Algorithms were developed based on R-join (reachability join) and they included optimisation techniques to optimise R-joins. In [58], a labelling schema was proposed to judge the reachability relationships between nodes of XML documents and two new structural join algorithms were developed based on that. Moreover, authors proposed sub-graph queries that are able to process queries with cycles.

The main focus of these studies was the case where XML elements transitively reference each other (without cycles) such as publications citing each other. Therefore, extra computations are required to find reachability between each two nodes which is handled by calculating the transitive closure of the correspondent graph [59]. However, it is not often the case where there are transitive references between XML elements, which means that GPM techniques are not appropriate in this case because they produce significant overhead that is not required in addition to increasing the overall complexity of the system [60].

#### **3.4.4. XML Query Relaxation**

One of the earliest methods proposes for approximate query matching is Query Relaxation or Tree Pattern Relaxation (TPR) which is based on modelling a query as a tree. TPR is used to describe the process of generalising a Pattern Tree Pt so that it returns more results that do not fully match the structure of Pt. The most popular study in TPR was conducted by [14] which paved the way to many others. Relaxation can be any combination of the following techniques [14] as shown in figure16:

- Node generalisation: a node can be generalised using a type hierarchy e.g. a node “book” in a query can be generalised to a node “document” so that journal articles and other types of documents are also included.

- Edge relaxation: a parent-child edge between two nodes can be generalised into an ancestor descendant edge.
- Making a leaf node optional: a leaf node in Pt can be made optional so that if it does not have a matching node in the data source, Pt is still considered as matching.
- Sub-tree promotion: this is where a sub-tree (of Pt) is disconnected from its parent and connected (as a child) to the parent node of its parent.

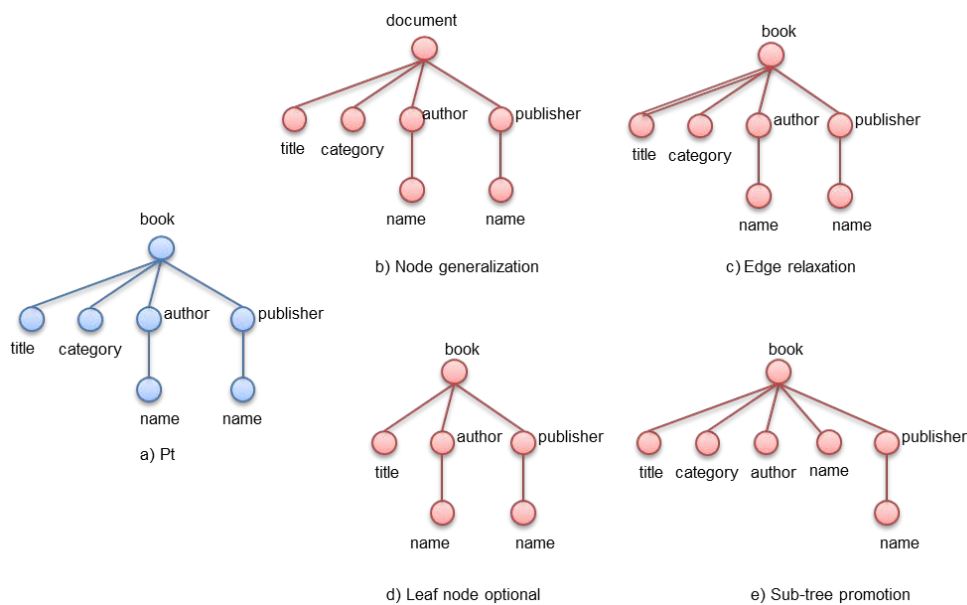


Figure 16: Query relaxation techniques

The more relaxations applied to a Pt, the more results it is likely to return.

However, those results returned through relaxations are *approximate* or

*similar* but not exact results. Authors proposed having weighted queries in order to measure the degree of relaxation. Each node and edge in Pt is assigned a weight and the total score for a Pt is the total of score of all of its nodes and edges. Users can define thresholds and only relaxed Pt's with higher score are considered. In a further study [15], Amer-Yahia et al. extended structural query relaxation by introducing *contains*-relaxation which is a relaxation of the *contains* predicate used for keyword search in XML documents. The study combined both structure and keyword search aiming at joining two major paradigms for XML querying, database-style querying and IR-style querying.

Even though query relaxation was adopted in some IR-systems, it turns out that it is not the best solution. According to [17], not all relaxations are appropriate for all pattern trees. For example, the sub-tree promotion in figure 9 (e) will result in confusion about the node promoted node *name* as it might be thought of as the publisher name instead of the author name. Additionally, blind relaxation i.e. a relaxation that is not based on knowledge of the underlying schema, results in a number of relaxed queries and each of these needs to scan the whole data set, which is inefficient.

In a recent work, Liu et al. [16] presented adaptive relaxation, a schema-aware approach that considers the schema of the data source before running queries against it. For a pattern tree, a set of relaxed queries is generated and

then compared against the schemas of underlying data sources. A relaxed query will be executed only if it satisfies the structural constraints imposed by the conformed schema, thus avoiding blind relaxation.

More studies on improving query relaxation were carried out. Fuzzinga et al. [18] extended previous techniques by introducing textual predicate relaxations such as *Relaxation of Equality Predicate* and *Predicate deletion* and used that to achieve approximate XPATH query matching. The proposed approach consisted of a new idea for combining partial answers from more than one data source and joining them based on key constraints. Authors claimed that their approach guarantees full automation as users do not need to be aware of underlying schemas and no mappings are provided. A different approach was presented by the same authors, Fazzinga et al., in [19, 20] where user requirements captured by a pattern tree were split into sub-patterns  $p_1, p_2 \dots p_n$  each represents a condition in the query. Un-matching sub-patterns are relaxed by replacing them with more relaxed ones so that more answers are retrieved. New relaxation techniques were proposed such as step cloning (duplicated a predicate on a node). In essence, Fazzinga's relaxations seem to be efficient especially that they were *schema-aware* relaxations.



### 3.4.5. Tree Algebra for XML (TAX)

Jagadish et al. [61] proposed TAX, an algebraic framework for XML query matching, which is an extension to relational algebra. Authors compared two approaches for XML query processing. The first works by transforming a collection of trees representing XML documents into a set of relational tuples, and then manipulating the resulting tuples using pure relational terms. Finally, answers are re-transformed into XML. However, this would need a lot of relational construction and deconstruction operations which will add significant overhead. Additionally, this approach does not leave an opportunity for query optimisation. The second approach is manipulating XML data as pure trees. This would avoid the pitfalls of the previous approach but it will face a challenge due to the heterogeneity in XML structures. TAX is proposed to address the challenges of the second approach.

A comprehensive set of XML operators were presented; mainly, selection, projection, and product (join), set operations, grouping, aggregation and others. The TAX selection operation is the analogue for relational selection which indicates that data trees that satisfy selection predicates specified in the pattern should be returned. Formally, a TAX selection is defined by  $\sigma_P, SL(C)$ . It takes a collection  $C$  of XML documents (trees) as input, a pattern tree  $P$  and an adornment list  $SL$  as parameters, and returns a collection of

trees as output. Each data tree in the output, called witness tree, induced by an embedding of  $P$  into  $C$ , modified as prescribed in  $SL$ . The adornment list  $SL$  lists the nodes from  $P$  for which all the descendants will be returned. This is a main extension of relational selection operation that applies only to XML due to its complex structure.

Projection is formally defined as  $\pi_{P, PL}(C)$ , it takes a collection  $C$  of XML documents (trees) as input, a pattern tree  $P$  and a projection list  $PL$  as parameters. A projection list  $PL$  consists of node labels appearing in the pattern  $P$ .

While selection and projection operations choose rows and columns in relation algebra, they are defined differently in TAX algebra in a way that makes them independent. Figure17 shows a collection of XML data trees  $C$ , a pattern  $P$  along with the result of a selection and projection operations on the collection. The selection list for the selection operation is assumed to be empty while the projection list  $PL$  for the projection operation consists of the nodes (book, author). In addition to selection and projection, authors presented details of other operators, but for the purpose of this thesis and because of space limitations, we only addressed the two main operators, selection and projection.

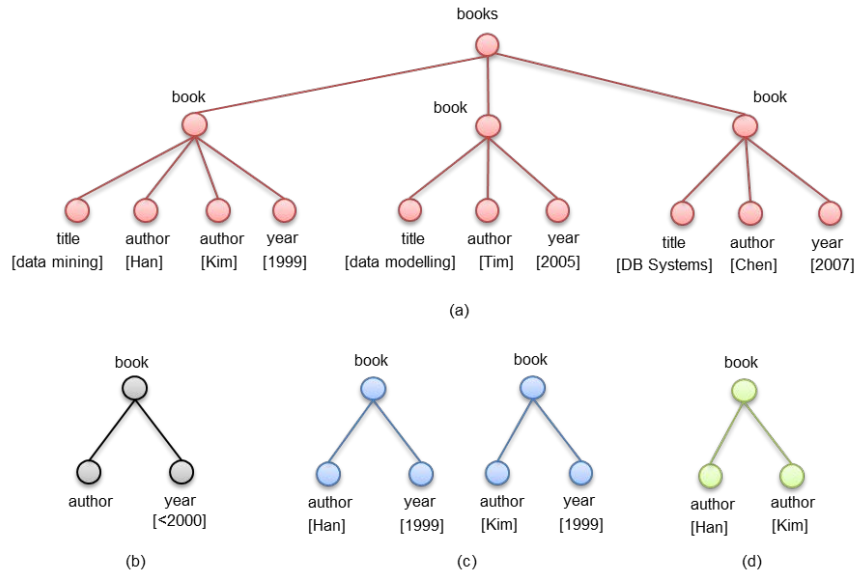


Figure 17: (a) collection C, (b) pattern P, (c) result of  $\sigma_{P, SL}(C)$ , (d) result of  $\pi_{P, PL}(C)$

In addition to defining TAX operators, authors defined Pattern Trees. A Pattern Tree was formally defined as a pair  $P=(T, F)$  where  $T=(V, E)$  is a noded and edged tree and  $F$  is a formula representing predicates (conditions) that apply to nodes. A Witness Tree was defined as the embedding of a pattern tree  $P$  into a collection  $C$  as a mapping  $h:P \rightarrow C$  from the nodes of  $T$  to those of  $C$  such that  $h$  reserves the structure of  $T$  i.e. all nodes and edges are matching and  $h$  satisfies the formula  $F$ .

### 3.4.6. Pattern Tree mining

Some researchers proposed a different approach for information extraction from heterogeneous XML data sources based on frequent pattern mining, which is an extension to the frequent itemset discovery problem in

association rule mining [12, 13, 62]. In [62], authors studied the problem of discovering frequent patterns (that have a minimum support) in a given collection of semi-structured (XML) data, which can be used for discovering structural patterns from large collections of semi-structured data (called semi-structured data mining). A pattern mining algorithm called FREQT was introduced where a technique called “right most expansion” was used to grow a tree by adding nodes to the rightmost branch only. Matching a pattern tree  $P_t$  to a schema tree  $S_t$  was defined based on matching the nodes of  $P_t$  to the nodes of  $S_t$  such that the parent child relations, sibling relations and node labels are preserved.

In [12, 13], an approach was proposed for mining XML mediator schemas from a set of heterogeneous XML databases, which they called “schema mining”. Frequent subtrees were extracted and then merged in order to build a mediator schema. The new idea in these works was Fuzzy Tree Inclusion, which means that a tree does not have to be fully included in another, it can be partially included. An algorithm was used to calculate the degree a Tree  $S$  is included in a tree  $T$  based on four parameters: a) Fuzzy vertical paths: a fuzzy approach was proposed to soften classical ones. If the number of nodes separating the ancestor from descendant is less or equal to 5 then  $S$  is considered as embedded in  $T$  with a degree from 0 to 1 depending on the number of separating nodes. b) Fuzzy horizontal paths: the proportion of nodes that are included and well-ordered is considered as the degree of

inclusion (from 0 to 1). c) Partial inclusion: if part of nodes from tree S is included in tree T then this proportion is related to a fuzzy quantifier which is then used as input to the function `Fuzzy_Inclusion_Degree`, a function that calculated inclusion degree. d) Similarity between nodes: fuzzy ontologies are used to specify to which extent two nodes are similar to each other depending on the semantics of their labels. In [13], authors introduced Fuzzy Links which provides more information about the link between two nodes whether it is very shared, middle shared, or little shared as shown in the figure18 below. The thicker the line (edge) connecting two nodes, the more it is common in data sources e.g. the edge (b, c) in the same figure, is the most common one.

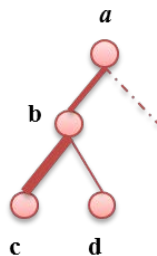


Figure 18:Fuzzy frequent subtrees

### 3.5. XML Query Rewriting

While most of the research on addressing structural heterogeneity in XML documents was focused on pattern tree matching, few studies suggested query rewriting mechanisms to overcome that heterogeneity. The most

popular one was conducted by Yu and Popa [63], where authors proposed a framework for answering queries through a *virtual* target schema. It was assumed that a set of formally defined mappings between source schemas and this target schema was provided, and that the data was indeed at the sources. It was also assumed that some constraints, such as key constraints, were defined on the target schema. Two algorithms were developed, a Query Rewriting Algorithm which rewrites the target query into a set of source queries based on the mappings; and a Query Resolution Algorithm that merges data from multiple sources by making use of the defined target constraints.

The proposed query rewriting algorithm consists of four phases i) Rule generation: creates a set of mapping rules based on the given mappings between the target schema and local data sources. ii) Query translation: it uses the rules defined from the previous phase to reformulate target queries into unions of source queries. iii) Query optimisation: unmatched source queries are removed and the matched ones are minimised, and iv) Assembly: reassembles decorrelated source queries back into queries with nested sub queries.

A similar solution was proposed in [64], but it was for query translation between P2P XML databases based on pre-defined informal mappings (linking arrows). Authors develop algorithms for inferring mapping rules

between schemas based on the provided informal mappings, as well as a language to express these mapping rules. The query translation algorithm can translate XML queries along and against the direction of mappings; and it is composed of four phases i) Expansion: a Pt is expanded so that it matches the mapping rules. Correspondences between original Pt and expanded Pts are kept track of. ii) Translation: expanded Pts are translated based on the mappings while keeping the query constraints. iii) Stitching: translated partial matchings are stitched together by making use of key constraints, and iv) Contraction: nodes that appear in the mapping rules but do not have matching nodes in Pts are called dummy nodes and are dropped from the translated query.

Overall, the pitfalls of the proposed algorithms in [63, 64] are that they are complex and they depends on manually defined mappings, which is not efficient when it comes to query matching.

Another interesting approach was introduced in [65]. Authors proposed a rewriting algorithm for integrated views over heterogeneous XML documents. XML schemas were modelled with ORA-SS model, a model that captures the semantic information of XML schemas. The proposed approach utilizes two tables: a mapping table that consists of mappings from an integrated schema to local schemas, and a query allocation table that stores the path information of XQuery selection and return parts. An algorithm was

developed to decompose the main query into a number of subqueries and join results from different local schema based on the information in the mapping and query allocation tables.

### **3.6. Chapter Conclusion**

In this chapter, a wide range of studies on XML schema matching/similarity and query matching were summarised and presented. Some of them addressed the structural heterogeneity of data sources, others focused on matching structure and content (query evaluation) aiming at achieving high performance and less use of memory.

The main focus of this chapter is on XML Query Matching studies, particularly the approximate structure-based matching, which is the subject of this thesis. Different approaches for tree similarity and inclusion were thoroughly discussed. TED is one of the earliest approaches used in IR communities for computing the similarity degree between two trees, one representing a query and the other representing a data source. Soon after, the terms Pattern Tree and Twig Patterns have become the most popular terms referring to graphic representations of queries over data trees.

Most of the research in that area was directed to query processing aiming at reducing access to data sources which in turn improves performance. Those studies assumed that schemas of underlying sources were already known,



therefore, they did not offer solutions to the structural heterogeneity problem. Other studies, however, proposed various approaches for structure-based pattern matching. Some of them were based on TED similarities; others considered the number of matching nodes as the basis of similarity. On the node (element) similarity level, some studies proposed techniques based on node labels, semantics, constraints and even positions in some cases.

A different solution to structural heterogeneity is query relaxation. This refers to a set of techniques used to generalise a query so that it retrieve more results that do not exactly match the structure of a query. Query relaxation has been widely adopted in IR systems especially for top-k queries.

Overall, the aforementioned studies provided techniques that solved certain types of structural diversity in data sources, but none of them proposed solutions for cases where data schemas are highly diverse. In particular, no studies addressed the case where two nodes in an XML schema are modelled as parent-child instead of child-parent, or where a node is modelled as an attribute instead of an element node. Furthermore, the proposed approaches for approximate matching of parent-child paths to ancestor-descendant do not always provide correct results. In addition, no solutions were proposed for efficient matching of a pattern against normalised XML documents i.e. documents with ID/IDREF connections.

## **Chapter Four: Intuitionistic Fuzzy Trees**

### **4. Intuitionistic Fuzzy Trees**

#### **4.1. Intuitionistic Fuzzy Logic (IFL)**

#### **4.2. Intuitionistic Fuzzy Trees (IFT)**

## 4. Intuitionistic Fuzzy Trees

In this chapter we present Intuitionistic Fuzzy Trees (IFT) which extends previous works on Intuitionistic Fuzzy Graphs. We redefine support and confidence, which are used in association rule mining to reflect frequent itemsets, to represent the degree of structural similarity/inclusion between two trees. Moreover, we present a novel algorithm for calculating these similarity measures.

### 4.1. Intuitionistic Fuzzy Logic (IFL)

Before introducing fuzzy logic, it is necessary to start from the classic or binary logic which consists of two values, True and False (or 1 and 0 respectively) e.g. a variable such as *pass* can be given a value of 1 (success) or 0 (failure). Fuzzy logic, however, allows variables to have values ranging from zero to one (0-1). Firstly introduced by Zadeh[66], fuzzy logic can be used for linguistic variables i.e. variables whose values are words in a natural language e.g. age, beauty, height etc. For example, a person who is 25 years old can have the variable *young*=0.80 whereas someone who is 30 years old would probably have *young*=0.50.

Zadeh[66]states that a fuzzy subset  $A$  of a universe of discourse  $U$  is characterized by a membership function  $\mu_A: U[0, 1]$  which associates

with each element  $u$  of  $U$  a number  $\mu_A(u)$  in the interval  $[0, 1]$ , with  $\mu_A(u)$  representing the grade of membership of  $u$  in  $A$ .

Building on Zadeh's work, Atanassov presented an extension to fuzzy sets he called Intuitionistic Fuzzy Sets (IFS)[67]. In addition to a degree of membership  $\mu(x)$ , IFS elements consist of functions for the degree of non-membership  $\nu(x)$  and indeterminacy/uncertainty  $\pi(X)$ . Thus, an IFS is defined as follows:

Let a set  $E$  be fixed. An IFS (Intuitionistic Fuzzy Set)  $A$  in  $E$  is an object of the following form:

$$A = \{ \langle x, \mu_A(x), \nu_A(x) \rangle \mid x \in E \}$$

Where functions  $\mu_A: E \rightarrow [0, 1]$  and  $\nu_A: E \rightarrow [0, 1]$  determine the degree of membership and the degree of non-membership of the element  $x \in E$ , respectively, and for every  $x \in E$ :

$$0 \leq \mu_A(x) + \nu_A(x) \leq 1$$

To make it clearer, suppose that we have  $E$  as the set of presidential candidates (Bush, Obama) and we define  $\mu()$  as people who voted for the candidate Bush and  $\nu()$  as people who voted for someone else. Suppose that 30% of the voters voted for Bush and 55% voted for Obama. This means that:

$$\mu(\text{Bush}) = 0.30 \text{ and } v(\text{Bush}) = 0.55$$

Someone might argue that  $v(\text{Bush})$  should be 0.70 instead of 0.55 which is not right because  $v(\text{Bush})$  represents the percentage of people who voted for someone else rather than Bush. It is known for sure that 55% voted for Obama but the missing 15% (i.e.  $1 - 0.30 - 0.55 = 0.15$ ) did not vote for Obama, those might have voted with blank or invalid forms. Thus the 15% are referred to as  $\pi(\text{Bush})$  i.e. the uncertainty of people voting for Bush.

## 4.2. Intuitionistic Fuzzy Trees (IFT)

Intuitionistic Fuzzy Graphs (IFG) was first introduced by Shannon and Atanassov in 1994 [68]. As a Tree is a special case of a Graph, the concept of IFT is defined as a restriction on the IFG [69-71]. Below we introduce a number of definitions to illustrate the IFT properties as by [72].

### Definition 1: Intuitionistic Fuzzy Graphs (IFG)

Let the oriented graph  $G = (V, A)$  be given, where  $V$  is a set of vertices and  $A$  is a set of arcs. Every graph arc connects one or two graph vertices.

$$A^* = \{ \langle \langle v, w \rangle, \mu_A(v, w), \nu_A(v, w) \rangle \mid \langle v, w \rangle \in V \times V \}$$

The set  $A^*$  is called an IFG if the functions  $\mu_A: V \times V \rightarrow [0, 1]$  and  $\nu_A: V \times V \rightarrow [0, 1]$  define the respective degrees of membership and non-membership of the element  $\langle v, w \rangle \in V \times V$  and for all  $\langle v, w \rangle \in V \times V$ :

$$0 \leq \mu_A(v, w) + \nu_A(v, w) \leq 1$$

**Definition 2: Intuitionistic Fuzzy Trees (IFT)**

An IFT is a restricted form of an IFG, with additional features. Same as the difference between traditional graphs and trees, IFTs are directed IFGs with parent-child connections and no cycles, where there is no more than one parent for each child. Additionally, the membership ( $\mu$ ), non-membership ( $\nu$ ) and hesitation ( $\pi$ ) functions of IFGs are also adopted in IFTs, along with additional features. Fuzzy Support and Fuzzy Confidence measures (See definition 5) have been introduced in IFTs to indicate to the degree of membership (Belief) and non-membership (Disbelief) of a tree being similar to or included in another tree.

Let  $V$  be a fixed set of vertices. Given that  $(v \in V)$  and  $(A \subset V \times V)$ , An IFT  $T$  over  $V$  will be the ordered pair  $T = (V^*, A^*)$ , where

$$V^* = \{\langle v, \mu_v(v), \nu_v(v) \rangle \mid v \in V\}$$

$$A^* = \{\langle g, \mu_A(g), \nu_A(g) \rangle \mid (\exists v, w \in V)(g = \langle v, w \rangle) \in A\}$$

Where  $\mu_v(v)$  and  $v_v(v)$  are degrees of membership and non-membership of the element  $v \in V$  and

$$0 \leq \mu_v(v) + v_v(v) \leq 1.$$

$\mu_A(g)$  and  $v_A(g)$  are degrees of membership and non-membership of the element

$$g = \langle v, w \rangle \in A \text{ and } 0 \leq \mu_A(g) + v_A(g) \leq 1$$

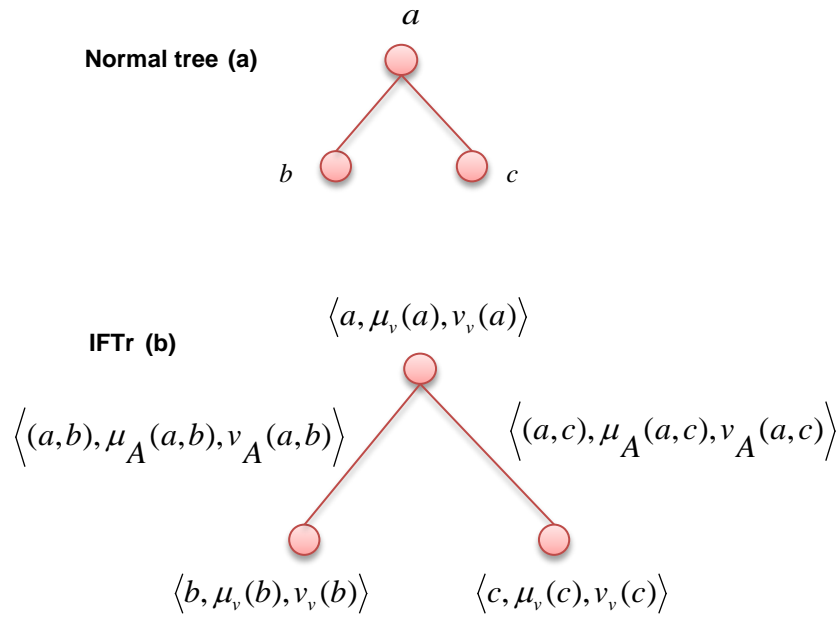


Figure19: (a) Normal tree vs (b) IFT

To clarify the definition of IFT, figure 19 shows a normal tree along with the correspondent IFT. In addition to the node labels, the IFT has functions that define the degree of membership and non-membership of each element of

the tree into another tree. For calculating IFT Inclusion we present a set of properties.

- If  $n$  is a node in a tree  $T$  then  $label(n)$  is a function that defines the label of  $n$ .
- If  $m, n$  are nodes in  $T$  such that  $n$  is a child of  $m$ , then  $A(m, n)$  will be the arc connecting  $m$  to  $n$ .
- $N(T)$  = a set of all nodes in  $T$
- $A(T)$  = a set of all arcs in  $T$
- $parent(n)$  = the parent node of  $n$
- $children(n)$  = the child nodes of  $n$
- $desc(n)$  = the descendant nodes of  $n$
- $anc(n)$ : the ancestor nodes of  $n$
- $\perp$  = Null

### Definition 3: Full Tree Inclusion

Let  $T_1$  and  $T_2$  be labeled trees. We define Full Tree Inclusion  $(\mathcal{O}, T_1, T_2)$  as a function  $\mathcal{O}: N(T_1) \rightarrow N(T_2)$  such that for all nodes  $m, n \in N(T_1)$

- $label(n) = label(\mathcal{O}(n))$
- $A(m, n) = A(\mathcal{O}(m), \mathcal{O}(n))$

### Definition 4: Partial Tree Inclusion



Let  $T_1$  and  $T_2$  be labeled trees. We define Partial Tree Inclusion  $(\mathcal{O}, T_1, T_2)$

as a function  $\mathcal{O}: N(T_1) \rightarrow N(T_2)$  such that for all nodes  $m, n \in N(T_1)$ ,

- $label(n) = label(\mathcal{O}(n))$  or  $\mathcal{O}(n) = \perp$
- $A(m, n) = A(\mathcal{O}(m), \mathcal{O}(n))$  or  $A(\mathcal{O}(m), \mathcal{O}(n)) = \perp$

In other words,  $T_1$  can be partially included in  $T_2$  when there are some nodes (and arcs) of  $T_1$  that do not exist in  $T_2$ .

### Definition 5: Support and Confidence

The degree of inclusion of a tree  $T_1$  in another tree  $T_2$  is  $\delta(T_1, T_2)$ . We define two factors that determine to which degree  $T_1$  is included in  $T_2$ :

- Support (S) = (# of nodes in  $T_1$  that are included in  $T_2$ ) /  $|T_1|$
- Confidence (C) = (# of arcs in  $T_1$  that are included in  $T_2$ ) /  $|T_1|$

Such that:  $|T_1|$  is the size (number of nodes) of  $T_1$ .

In other words, *Support* represents the percentage of nodes in  $T_1$  that are included in  $T_2$  individually (on the element level) without considering the node position (structure) whereas *Confidence* represents the percentage of nodes in  $T_1$  that are included in  $T_2$  in the right structure (on the structure level). The basic unit of structure that is considered here is arc, which connects two nodes with a parent-child relationship. Therefore, the total number of included (matched) arcs is considered as part of the measure

Confidence (C). For calculation precision, the value 1 is added to the number of matching arcs and the total is divided by  $|T_1|$ . This is because (the number of matching arcs + 1) equals the number of matching nodes (considering the position/structure of the nodes). For example, for the two trees,  $T_1$  and  $T_2$  shown in figure 13, the inclusion of  $T_1$  into  $T_2$ ,  $\delta(T_1, T_2)$ , can be calculated by finding S and C as follows:

$S=5/5=1$  (100%) which means that all nodes of  $T_1$  are individually included in  $T_2$

$C=4/5=0.8$  (80%) which means that 4 out of 5 nodes in  $T_1$  are included and structured properly in  $T_2$ . These are  $\{b, c, d, e\}$ . Notice that even though the node  $a$  is included in  $T_2$ , it is not structured properly. In  $T_1$ ,  $a$  is the parent of  $b$ , whereas in  $T_2$   $a$  is a sibling of  $b$ .

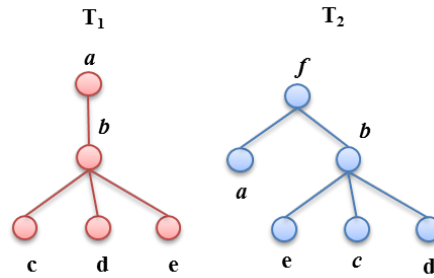


Figure 20: Tree inclusion of  $T_1$  into  $T_2$

### Definition 6: Belief, Disbelief and Hesitation

Based on the previous terms, S and C, we define the followings:

- Belief ( $\mu$ ) = belief of  $T_1$  being included in  $T_2$ . S.t.  $\mu = C$
- Disbelief ( $\nu$ ) = disbelief of  $T_1$  being included in  $T_2$ . S.t.  $\nu = 1 - S$
- Hesitation ( $\pi$ ) = hesitation of  $T_1$  being included in  $T_2$ . S.t.  $\pi = S - C$
- Maximum belief ( $\mu_{\max}$ ) = the maximum believe of  $T_1$  being included in  $T_2$ .  $\mu_{\max} = C + \pi$
- Believe ( $\mu$ ) + Disbelief( $\nu$ ) + Hesitation ( $\pi$ ) = 1

**Definition 7: Node and Arc similarity functions**

- $sim(n, n')$  = Similarity between two node labels,  $label(n)$  and  $label(n')$ , which has a value range from  $[0,1]$ . If labels are identical, then  $sim(n, n')=1$ , else  $sim(n, n')$  determines the similarity between the semantics of the two labels. This can be obtained by using a semantic lexicon such as WordNet [73] or Linguatools [74]. Overall,  $sim(n, n')$  is equivalent to the membership function  $\mu_v(n)$ , which indicates to the degree of belief that  $n$  is similar to  $n'$ .

For example, suppose that  $label(n)$  = “quantity” and  $label(n')$  = “amount”, by using the semantic lexicon Linguatools [74],  $sim(n, n')=0.72$  which means that the semantics of the two labels “quantity” and “amount” are relatively close.

- $sim(A(m, n), A(m', n')) = \text{Similarity between two arcs } A(m, n) \text{ and } A(m', n')$  ranges from  $[0,1]$  which is equivalent to  $\mu_A(A)$ .

**Definition 8: Intuitionistic Fuzzy Support and Confidence**

- Intuitionistic Fuzzy Support ( $S_f$ ): for every node  $n \in N(T_1)$  and its correspondent node  $n' \in N(T_2)$

$$(S_f) = \sum \mu_v(n) / |T_1|$$

Where  $\mu_v(n)$  is the maximum degree of similarity (membership) between  $n$  and  $n'$ .

- Intuitionistic Fuzzy Confidence ( $C_f$ ): for every arc  $A(m, n) \in A(T_1)$  and  $A(m', n') \in A(T_2)$

$$(C_f) = (\sum \mu_A(A) + 1) / |T_1|$$

Where  $\mu_A(A)$  is the maximum degree of similarity (or membership) of an arc  $A(m, n) \in A(T_1)$  and  $A(m', n') \in A(T_2)$ .

To calculate the composite similarity measure  $\langle S_f, C_f \rangle$ , an IFT Inclusion Algorithm was developed (Figure 21)[75, 76]. It takes two trees  $T_1$  and  $T_2$  as input and calculates  $S_f$  and  $C_f$  as output. The algorithm calls a function called `mapNodes()` which iterates through nodes of  $T_1$  comparing each against nodes of  $T_2$  and resulting in a Node Mapping Matrix (NMM). In this

process, each node from T1 is mapped to a node (or more) from T2 based on label or semantics similarity provided that the similarity exceeds a predefined threshold. The algorithm iterates through node mappings and considers the node mapping with the highest similarity for calculating Sf.

The same applies to arc matching when calculating Cf. An Arc Mapping Matrix (NMM) is returned by the function mapArcs(), which compares arcs of T1 to those of T2. Again, an arc A(m, n) from T1 can have one or more matching arcs in T2, probably with different matching degrees. An arc matching threshold can be used to filter out weak arc mappings. In case of one-to-many arc mapping, the IFT Inclusion Algorithm considers the mapping with the highest arc similarity for calculating Cf.

### IFT Inclusion Algorithm

// This algorithm calculates Sf and Cf which imply the //inclusion degree of T1 in T2.

Input: Two trees T1 and T2, NMM and AMM

Output: Sf, Cf

Begin

Sf=Cf= matchedNodes= matchedArcs=0; //initialization

// calculate Sf

mapNodes(T1, T2); // generates Node Mapping Matrix (NMM)

For each  $n \in N(T1)$ {

    maxSim=0;

    For each  $m \in N(T2)$  such that  $n \rightarrow m \in NMM$ {

        If  $\text{sim}(n, m) > \text{maxSim}$

            maxSim= $\text{sim}(n, m)$ ;

    }

    matchedNodes+=maxSim;

}

Sf = matchedNodes / |T1|;

// calculate Cf

mapArcs(T1, T2); //generates Arc Mapping Matrix (AMM)

For each Arc  $A(m,n)$  in T1

    maxSim=0;

    For each  $A(m',n')$  in T2 such that  $A(m,n) \rightarrow A(m',n') \in AMM$ {

        If  $\text{sim}(A(m,n), A(m',n')) > \text{maxSim}$

            maxSim= $\text{sim}(A(m,n), A(m',n'))$ ;

    }

    matchedArcs+=maxSim;

}

Figure 21: IFT Inclusion Algorithm

## **Chapter Five: Intuitionistic Fuzzy Pattern Tree Matching**

### **5. Intuitionistic Fuzzy Pattern Tree Matching**

#### **5.1. Overview**

#### **5.2. Soft Node Matching**

#### **5.3. Soft Arc Matching**

##### **5.3.1. Direct Match**

##### **5.3.2. Inverted Match**

##### **5.3.3. AttNode Match**

##### **5.3.4. Normalized Match**

##### **5.3.5. Separating Node Match**

##### **5.3.6. Hybrid Arc Match**

#### **5.4. Pattern Tree Matching Matrices**

##### **5.4.1. Node Mapping Matrix (NMM)**

##### **5.4.2. Arc Mapping Matrix (AMM)**

##### **5.4.3. Query Index Matrix (QIM)**

#### **5.5. Chapter Conclusion**

## 5. Intuitionistic Fuzzy Pattern Tree Matching

In this chapter, the IFT approach is applied to Pattern Tree Matching. An illustrative example is given in section 5.1 to further explain the benefits of the proposed approach. Soft node and arc matching techniques are presented in sections 5.2 and 5.3 along with explanatory definitions. Section 5.4 introduces the matrices required to hold node and arc matching results.

### 5.1. Overview

The additional benefit of using IFT is that it gives more information about how much a pattern tree Pt matches an underlying schema tree St. It provides the confirmed minimum degree to which Pt is included in St ( $C_f$ ), the maximum degree of inclusion in the best case ( $S_f$ ), the degree of exclusion ( $1 - S_f$ ) and the hesitation ( $\pi$ ) which implies to which extent we are not sure that Pt is included in St.

To clarify the above, we calculate  $S_f$  and  $C_f$  for Pt in St1 and St2 (figure 22). Let  $S_{f1}$  and  $C_{f1}$  denote the Intuitionistic Fuzzy Support and Confidence for St1, respectively;  $S_{f2}$  and  $C_{f2}$  denote the same for St2. Obviously,  $S_{f1}$  is 1.0 as all the nodes of Pt are included in St1. However,  $S_{f2}$  will be less than 1.0 as the node *pname* in Pt does have a match. Therefore:

$$S_{f2} = (\# \text{ of nodes in Pt that are included in St}_2) / |\text{Pt}|$$



$$= 5/7 = 0.71$$

$C_{f1}$  ( $\mu_1$ ) and  $C_{f2}$  ( $\mu_2$ ) are calculated as follows: (See definition

$$C_{f1} = (\# \text{ of arcs in } Pt \text{ that are included in } St_2 + 1) / |Pt|$$

$$= 5 + 1/7 = 0.86$$

$$C_{f2} = 4 + 1/7 = 0.71 \text{ (See algorithm in figure 21)}$$

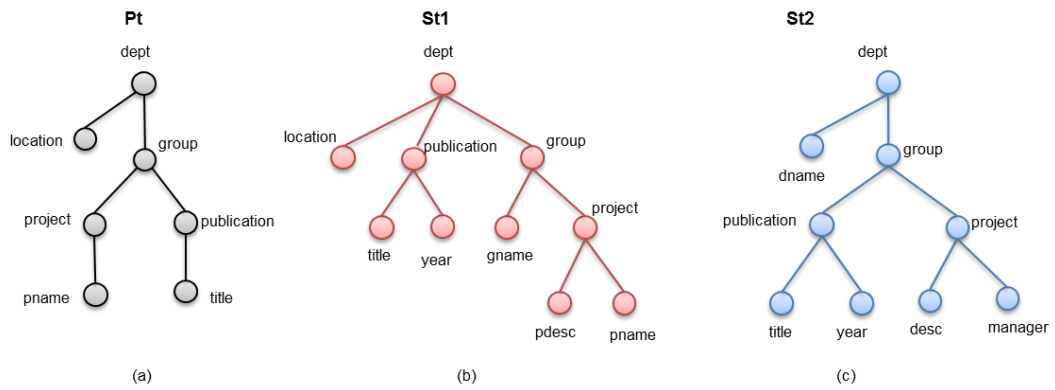


Figure 22: A pattern tree with two different schema trees

Having calculated the support and confidence, the hesitation ( $\pi$ ) of considering  $Pt$  included in  $St_1$  and  $St_2$  can now be calculated as follows:

$$\pi_1 = S_{f1} - C_{f1}$$

$$= 1.0 - 0.86$$

$$= 0.14 \text{ i.e. there is hesitation of considering 14\% of } Pt \text{ included in } St_1$$

$$\pi_2 = S_{f2} - C_{f2}$$

$$= 0.71 - 0.71$$

= 0.0 i.e. there is no hesitation of considering some part of Pt included in St2

Also the disbelief ( $v$ ) can be calculated as follows:

$v_1 = 1 - S_{f1} = 0$  i.e. there is no confirmation that some part of Pt is not included in St1

$v_2 = 1 - S_{f2} = 0.29$  i.e. 29% of Pt is certainly not included in St2.

As shown from the calculations, Pt is more included in St1 than of St2. Sf is high in both of them, however  $C_f$  achieved different results. 5 out of 6 arcs of Pt were included in St1 which scores high confidence indicating that Pt is included in St1 with 86% belief. The score of  $C_{f2}$ , on the other hand, was not high, which indicates that Tp is included in Td2 with 58% belief. The big difference between  $S_{f2}$  and  $C_{f2}$  causes hesitation ( $\pi_2$ ) to be high (28%). Also the disbelief ( $v$ ) can be calculated to indicate to which degree Tp is NOT included in a data tree. While  $v_1$  is zero,  $v_2$  has a score of 14%, which indicates that 14% of Tp is certainly not included in Td2. Adding the measures of Td2 together will sum up to 1.0.

$$\mu_2 + v_2 + \pi_2 = 0.58 + 0.14 + 0.28 = 1.0$$

By using three values to calculate the level of matching between a query and a set of XML schema trees, IFT has the ability to provide more information on the matching degree than previous works do, which is expected to return better query answers.

Once the degree of inclusion is calculated by finding  $\langle Cf, Sf \rangle$ , if the degree of inclusion is higher than a predefined threshold then the schemas are semantically close and Pt counts as a *witness tree*. The reason of using Intuitionistic Fuzzy techniques is to soften the traditional constraints on finding the degree of inclusion. The “source” tree does not need to be completely included in the “destination” one; it can be partially included.

Here we try to make it even more flexible by considering cases where nodes' labels and arcs that are not fully matching. Formally, if  $n$  and  $m$  are two nodes in Pt forming an arc  $A(m, n)$ , the correspondent Arc in St is  $A(m', n')$  such that

$$n \rightarrow n' \text{ and } m \rightarrow m' \text{ where the symbol '}\rightarrow\text{' reads “maps to”}$$

We propose two ways of softening matching rules: soft node matching and soft arc matching.

## 5.2. Soft Node Matching

An algorithm is developed to softly match nodes of a Pt with nodes of St's (figure23). Two nodes do not necessarily need to have the same label in order to be considered matching. If  $label()$  is a function that defines node labels then for each node  $n$ , it is not necessary that:

$$label(n) = label(n')$$

A linguistic (lexical) ontology is utilised to add semantics to node labels and then a function is invoked to compare these semantics and calculates the similarity (Semantic closeness) according to the distance between them. This is defined as:

$$semantics(n) \approx semantics(n') \text{ where the symbol } '\approx' \text{ reads "close to"}$$

Additionally, any two nodes can be matched together even if they do not have the same node type. Stated differently, an *element* node in a Pt can be matched to an *attribute* node in St provided that the element node is a leaf node (See AttNode arc matching in Figure24).

Pt nodes can be classified into different types according to their role in the query tree or the schema tree. In addition to element nodes (e.g. *dept*) and attribute nodes (e.g. *@dname*) in figure 24 (d), we define the following node types: Pattern nodes, Schema nodes, ID nodes, Output nodes, Intermediate nodes and Join nodes.

### Soft Node Matching Algorithm

Input: Two trees Pt and St, Node similarity threshold  $\Theta_N$

Output: Node Mapping Matrix (NMM)

Begin

For each node n in Pt {

For each node m in St {

    If label(n)= label(m)

        addNodeMapping(n, m, 1) //add n and m to NMM, 1 is for exact match

    Else If semSim(label(n), label(m))  $>\Theta_N$

        addNodeMapping(n, m, semSim(label(n),label(m)))

    }

}

End

**Function semSim (label(n), label(m))**

It calculates the semantic similarity between labels of nodes n and m by mapping the labels to a linguistic ontology (WordNet) and then calculating the distance between them according to the shortest path connecting them within the taxonomy.

Figure 23: Soft Node Matching Algorithm

#### Definition 9: Pattern Node

A pattern node n is any node within a pattern tree Pt i.e.  $n \in N(Pt)$

#### Definition 10: Schema Node

A schema node n' is any node within a schema tree St i.e.  $n' \in N(St)$

**Definition 11: ID Node**

A node can be an ID node if it can uniquely identify any instance of its parent node. For optimum query matching results, each parent node in Pt has to have an ID node. The reason is that it enables joining sub-trees from different schemas based on the ID of the common node. The labels of ID nodes are underlined in pattern trees to signify their role as a ‘Primary key’ of their parents.

**Definition 12: Output Node**

An Output node is a node whose value is to be returned in the query. It is distinguished by having a shape of double circles in pattern trees. An output node is either a leaf element node or an attribute node.

**Definition 13: Intermediate Node**

A node is said to be intermediate if it is neither a root node nor a leaf node i.e. it is a node that has a parent node and one or more child nodes.

**Definition 14: Join Node**

An intermediate node that has a child ID node and that is used to join two twigs together.

### Definition 15: PtNodes Matrix

The Pt Nodes Matrix is a matrix that consists of all node of a pattern tree Pt and all information about these nodes including: node labels, parent-child relationships, node type and role in Pt.

### 5.3. Soft Arc Matching

As arc is the fundamental unit of structure in data schemas, we propose different ways of approximate matching of a pattern arc with a schema arc. The main idea is to adapt to the different ways of modelling a parent-child relationship in different data sources. Figure 24 shows six different ways of matching the arcs (*group*, *publication*) and (*dept*, *dname*).

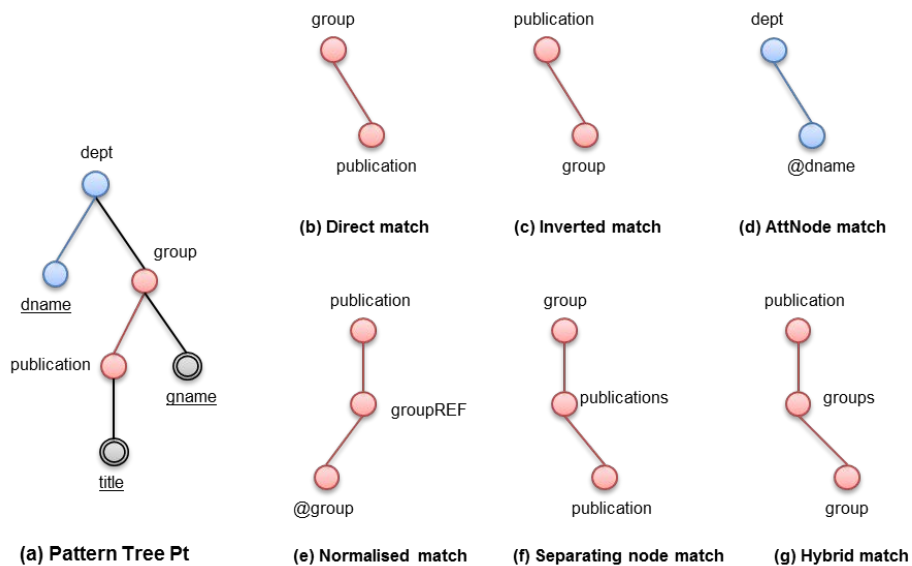


Figure 24: Types of soft arc matching

Before we proceed to the discussion of these types, we present few definitions on types of arcs: Leaf arc, Non-leaf arc, Pattern arc and Schema arc.

**Definition 16: Leaf Arc**

A leaf arc is an arc whose child node is a leaf e.g.  $A(dept, dname)$  in Figure 24 (a).

**Definition 17: Non-leaf Arc**

A non-leaf arc is an arc whose child node is not a leaf e.g.  $A(group, publication)$  in Figure 24 (a).

Depending on whether the arc is a leaf or non-leaf arc, different arc matching techniques (types) apply. In the following sections, we present different types of arc matching along with formal definitions and we explain to which types of arcs they apply.

**Definition 18: Pattern Arc**

A pattern arc  $A(m, n)$  is an arc within a pattern tree  $Pt$  i.e.  $A(m, n) \in A(Pt)$ .

**Definition 19: PtArcs Matrix**

PtArcs is a matrix of all pattern arcs.



**Definition 20: Schema Arc**

A schema arc  $A(m', n')$  is any arc within a schema tree  $St$  i.e.  $A(m', n') \in A(St)$

**Definition 21: ID Arc**

An ID arc is an arc whose child node is an ID node.

The following subsections present different types of soft arc matching as published in [77]. A proposed value of membership ( $\mu_A$ ) is presented and values for non-membership ( $\nu_A$ ) can be calculated by this formula:

$$\nu_A = 1 - \mu_A$$

**5.3.1. Direct Match**

In this type of match, a pattern arc  $A(m, n)$ , where  $m$  is the parent of  $n$ , is matched to an identical schema arc  $A(m', n')$  where  $m \rightarrow m'$  and  $n \rightarrow n'$ . This type of matching applies to both leaf arcs and non-leaf arcs which means that  $n$  can be either an element or an attribute node. Figure 24(b) shows an arc matched in that way. Formally, direct arc match is defined as:

$$A(m, n) \rightarrow_D A(m', n') \text{ iff } m \rightarrow m' \text{ and } n \rightarrow n' \text{ and } n'.parent() = m'$$

Since this is an exact match,  $\mu_A$  will be equivalent to 1.0.

### 5.3.2. Inverted Match

Unlike direct match, inverted match occurs when a pattern arc  $A(m, n)$  maps to a schema arc  $A(m', n')$  where  $m \rightarrow n'$  and  $n \rightarrow m'$ . This mismatch of modelling a relationship between two nodes  $m$  &  $n$  is common in XML documents depending on the modeller's perception, or point of interest. Arcs matching in this way should be non-leaf arcs because a leaf node cannot be modelled as child in one tree and as parent in another e.g. the arc  $A(dept, dname)$  in figure 24 (a) cannot be found as  $A(dname, dept)$  because the node  $dname$  is a leaf node that is correspondent to a text XML element, which cannot have children. However, if the arc is a non-leaf arc, such as  $A(group, publication)$  in figure 24 (a), it can be modelled as  $A(publication, group)$  in other schemas such as in figure 24 (c). Formally, inverted arc match is defined as:

$$A(m, n) \rightarrow_{\iota} A(m', n') \text{ iff } m \rightarrow n' \text{ and } n \rightarrow m'$$

Since this is not an exact match,  $\mu_A$  will be less than 1.0. For the proposed approach, 10% of belief is deducted as a result i.e.  $\mu_A=0.9$  to distinguish this match from Direct match. The 10% penalty is user defined and therefore it can be modified according to user's estimation.

### 5.3.3. AttNode Match

In relational databases, an entity type such as *dept* (department) has attributes such as *name*, *location* etc. In XML, however, a department name can be modelled either as an attribute node(*@dname*) or an element node (*dname*). However, since attribute nodes cannot have children, this type of arc match only applies to leaf arcs such as the one shown in figure 24 (d). Formally, AttNode arc match is defined as:

$$A(m, n) \rightarrow_A A(m', @n') \text{ iff } m \rightarrow m' \text{ and } n \rightarrow @n'$$

Again, 10% of belief is deducted as a result non exact match. Thus,  $\mu_A=0.9$ .

#### 5.3.4. Normalized Match

This match can be found in cases where an XML document is normalized i.e. each entity is modelled as a sub-tree (twig) within the document which can be referenced by using IDREF instructions. This can be thought of as analogy of the primary and foreign keys in relational modelling. To return data from more than one twig, an XML query joins the correspondent twigs based on a common node while having an attribute or element that acts as the ID of the common node. In Figure 25 the node *pubREF* is a reference node that refers to *publication* node within the same document. Consequently, we can say that the pattern arc (*group*, *publication*) is matching with (*group*, *pubREF*) using normalized arc match. Obviously, arcs matched in this way

should be non-leaf arcs because the child node (e.g. publication) should have a child ID node on which the join will take place.

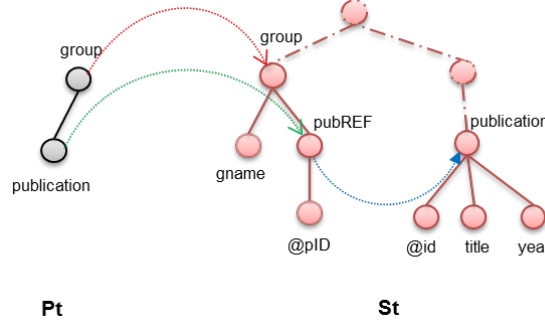


Figure 25: Normalised arc match

This type of match is complicated and it requires more processing resources i.e. time and memory. This is mainly because it is not enough to achieve this type of match based on the data schema only; the actual XML data document is also required because it is not possible to know to which element an IDREF attribute is referring without traversing the actual XML document. For our approach, we use a function that picks an instance of an IDREF element, such as *pubREF*, from the XML document, and scans it to find the parent of that ID e.g. *publication* node in our example. Formally, normalised arc match is defined as:

$$A(m, n) \rightarrow_N A(m', n') \text{ iff } m \rightarrow m' \text{ and } n \rightarrow \text{de-ref}(n')$$

Where  $\text{de-ref}(n')$  is a function that returns the node referenced by  $n'$ .

Same as previous types of approximate matching, 10% of belief is deducted making  $\mu_A=0.9$ .

#### 5.3.5. Separating node (SepNode) match

In some cases, when trying to match  $A(m, n)$ , an exact match might not be possible because of having separating nodes between the parent and the child. In this case the arc still can be matched if the number of separating nodes does not exceed a predefined threshold. Formally,  $A(m, n)$  can be matched with  $A(m', n')$  using this method if  $m$  is an ancestor of  $n$ , not necessarily a parent.

This matching has been proposed by many previous studies [17, 29]. However, it can result in getting the wrong result especially that this type applies to both leaf and non-leaf arcs. For example, suppose that we have a leaf arc  $A(\text{dept}, \text{location})$ , that is to be matched with a schema as in figure 26. Clearly, there is no matching arc in  $St$  because the node *dept* does not have a child *location*. However, using the separating node arc match, *dept* has a descendant node called *location*, which means that the arc can be approximately matched. But the problem is that the node *location* does not refer to the location of department, it refers to the location of the project. Thus, blind approximate matching using this technique can return wrong matchings.

To solve this dilemma, we consider that intermediate nodes are *strong* nodes as they usually represent independent entities (concepts) in relational models. Leaf nodes, however, usually represent attributes of their parents, and therefore they are *weak* nodes. If an arc  $A(m, n)$  is to be matched,  $m$  is a strong node and  $n$  is a weak node, any separating nodes between them can induce weak semantics. In other words, the weak node is more likely to have strong semantics with its parent e.g. the node location in figure26 refers to its parent (project) but not to its anc (dept). On the other hand if both nodes ( $m$  and  $n$ ) are strong nodes e.g. (dept, publication), then an intermediate node, such as group in our example, is unlikely to affect the semantics. In the same example, the arc  $A(\text{dept}, \text{publication})$  in Pt can be matched against St even if there is a separating node (group). This can be explained because a department consists of research groups, and these have publications. Thus, we can say that those publications belong to the department.

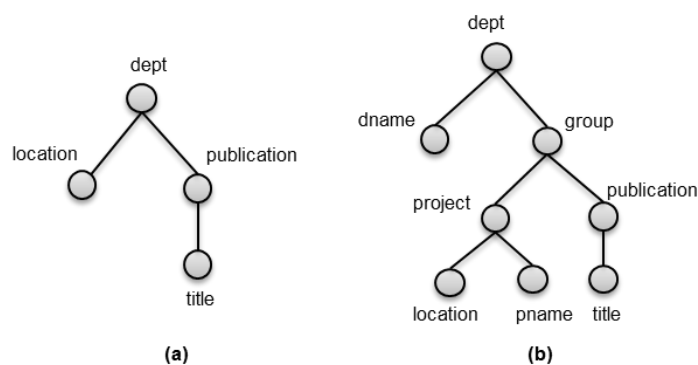


Figure 26(a) A pattern Pt, (b) a schema tree St

Formally, separating node match is defined as:

$$A(m, n) \rightarrow_s A(m', n') \text{ iff } m \rightarrow m' \text{ and } n \rightarrow n' \text{ and } m' \in \text{desc}(n') \text{ and } m' \text{ is not the parent of } n'$$

Unlike previous types of approximate matching, belief of this type depends on the number of separating nodes. Therefore, it is not correct to just deduct 10% of the belief; the amount deducted has to be proportional to the number of separating nodes. Belief of this type is defined by the following equation:

$$\mu_A = 1/(1+\partial/2) \text{ where } \partial \text{ is the number of separating nodes.}$$

The reason why  $\partial$  was divided by 2 is to reduce the effect of increasing number of nodes and make the belief reasonable.

### 5.3.6. Hybrid arc match

Not only can an arc be matched using the aforementioned approaches individually' but it can also be matched using combinations of them. Depending on the arc type, different combinations apply. For Leaf arcs, two types of approximate (indirect) match apply, AttNode and SepNode match. Thus, a Hybrid arc match for this type of arcs can be a combination of both AttNode and SepNode(AS) match. An example of this can be seen in figure27 where the pattern arc A(dept, location) is matched to the schema arc A(dept, @location) while having a separating node (info) that separates the

parent and child nodes. Formally, AttNode-SepNode hybrid match is defined as:

$$A(m, n) \rightarrow_{AS} A(m', n') \text{ iff } m \rightarrow m', n \rightarrow n', m' \in \text{anc}(n') \text{ and } n' \text{ is an attribute node}$$

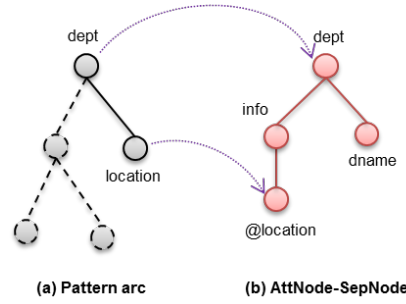


Figure 27: AttNode-SepNode hybrid match

Since the aforementioned type is hybrid, the amount deducted from belief is going to be the total of penalties applied for each type. Thus, the following equation defines belief for this type:

$$\mu_A = 90\% \times 1/(1+\partial/2) \text{ where } \partial \text{ is the number of separating nodes.}$$

The above equation is simply the same as of the one for SepNode match but multiplied by 90%. The 10% is the penalty for AttNode match.

Non-leaf arcs, on the other hand, can be matched using Inverted, SepNode, Normalised match or any combination of the three (or two) of them. Therefore, the following combinations can be found in a Hybrid arc match for non-leaf arcs: Inverted-SepNode (IS), Inverted-Normalised (IN),



SepNode-Normalised (SN) and finally Inverted-SepNode-Normalised (ISN) match. Figure28 shows examples of all of these combinations.

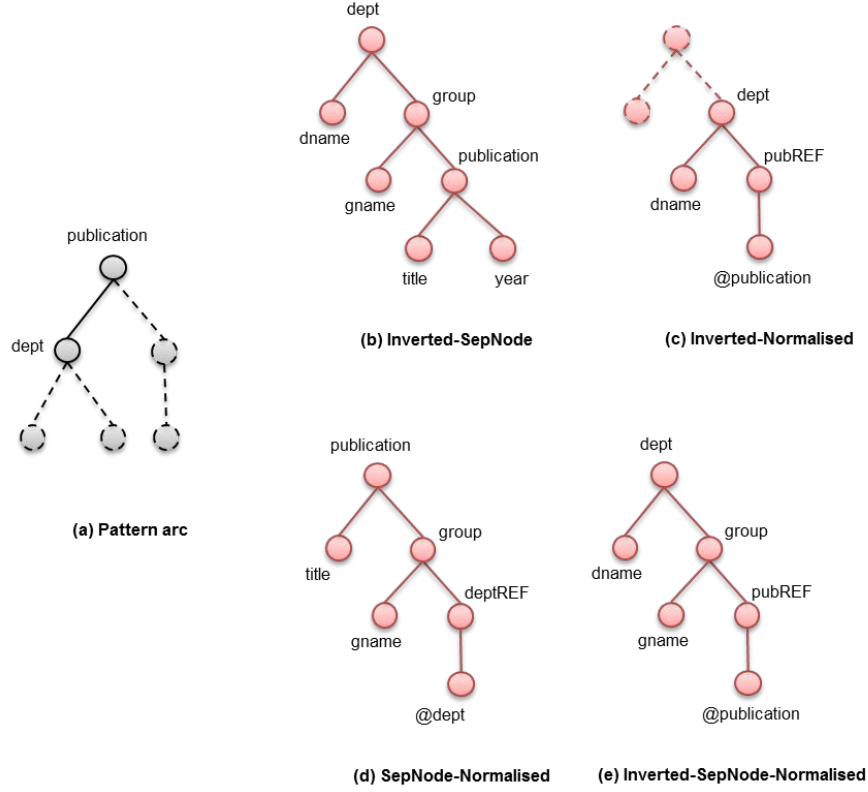


Figure 28: Different combinations of Hybrid arc match

The first combination is the Inverted-SepNode Hybrid (IS)match e.g. figure28- (b). In this type, a pattern arc  $A(m, n)$  is matched to a schema arc  $A(m', n')$  where  $m \rightarrow n'$  and  $n \rightarrow m'$  and  $n'$  is an ancestor (not parent) of  $m'$ . Formally, Inverted-SepNode hybrid match is defined as:

$$A(m, n) \rightarrow_{IS} A(m', n') \text{ iff } m \rightarrow n', n \rightarrow m', n' \in \text{anc}(m') \text{ and } n' \text{ parent}(m')$$

Similar to the AttNode-SepNode match, Inverted-SepNode belief is deducted twice as in the following equation:

$$\mu_A = 90\% \times 1/(1+\partial/2) \text{ where } \partial \text{ is the number of separating nodes.}$$

The second combination is the Inverted-Normalised Hybrid (IN) match e.g. figure 28-(c). There, an arc  $A(m, n)$  is matched inversely and the child node of the matching schema arc is a reference to the correspondent node. Stated formally, this type of match is defined as:

$$A(m, n) \rightarrow_{IN} A(m', n') \text{ iff } m \rightarrow \text{deref}(n'), n \rightarrow m' \text{ and } n' \neq \text{parent}(m')$$

Belief of Inverted-Normalised match is deducted twice, 10% penalty for each type in the hybrid match i.e.  $\mu_A = 80\%$ . The number of separating nodes is not considered here because it not applicable to this type.

The third combination is the SepNode-Normalised Hybrid (SN) match e.g. figure 28-(d) where a pattern arc  $A(m, n)$  is matched to a schema arc  $A(m', n')$  having  $m \rightarrow m', n \rightarrow \text{deref}(n')$  and  $m'$  is an ancestor but not parent of  $n'$ . Formally, this match is defined as:

$$A(m, n) \rightarrow_{SN} A(m', n') \text{ iff } m \rightarrow m', n \rightarrow \text{deref}(n'), n' \in \text{desc}(m') \text{ and } m' \neq \text{parent}(n')$$

The SepNode-Normalised match causes belief to be deducted twice as in the following equation:

$$\mu_A = 90\% \times 1/(1+\partial/2)$$

The last combination is the Inverted-SepNode-Normalised Hybrid (ISN) match e.g. figure28-(e) which consists of all types of individual arc matching types for non-leaf arcs. In this type, a pattern arc  $A(m, n)$  is matched to a schema arc  $A(m', n')$  where  $n \rightarrow m'$ ,  $m \rightarrow \text{deref}(n')$ ,  $m' \in \text{anc}(n')$  and  $m'$  is not  $\text{parent}(n')$ . Formally, this match is defined as:

$$A(m, n) \rightarrow_{ISN} A(m', n') \text{ iff } n \rightarrow m', m \rightarrow \text{deref}(n'), m' \in \text{asc}(n') \text{ and } m' \neq \text{parent}(n')$$

The ISN match consists of 3 types of anomalies; therefore its belief will be deducted three times as in the following equation:

$\mu_A = 80\% \times 1/(1+\partial/2)$  where the 80% represent the remaining after deducting 20% for Normalised and Inverted matching and the rest of the equation deducts from belief proportional to the number of separating nodes ( $\partial$ ).

Obviously, those combined hybrid matches are more complex to identify than individual ones and they might be subject to uncertainty in some cases. All the previous types of soft arc matching can be identified using a novel algorithm developed for that purpose as shown in figure 29.

Overall, the aforementioned types of matching, both individual and combined, are saved in several matrices so that they are used as input to the

query rewriting algorithm in order to rewrite the original query into a new one that is able to construct an answer based on the matchings. Next section presents the matrices used to process these matches.



Figure29: Soft Arc Matching Algorithm – part 1



Figure29: Soft Arc Matching Algorithm – part 2

## 5.4. Pattern Tree Matching Matrices

The result of matching a Pt against a number of St's is a set of mappings, node mappings and arc mappings. These are implemented using a number of matrices where each node and arc in Pt is correlated to its correspondent in St. figure30 shows the node mapping process where a one dimensional matrix (also a list can be used) is created for the Pt and for each St. The mapping cardinality between Pt nodes and St nodes is one-to-many e.g. in figure18, the node b in Pt is mapped to two nodes in St1 with the same label.

### 5.4.1. Node Mapping Matrix (NMM)

Node mapping between a pattern tree Pt and a schema tree St is defined as a function  $\phi_N(Pt, St)$  such that:

$$\phi_N(Pt, St) = \langle (n, m) | n \in Pt, m \in St \text{ and } n \rightarrow m \rangle$$

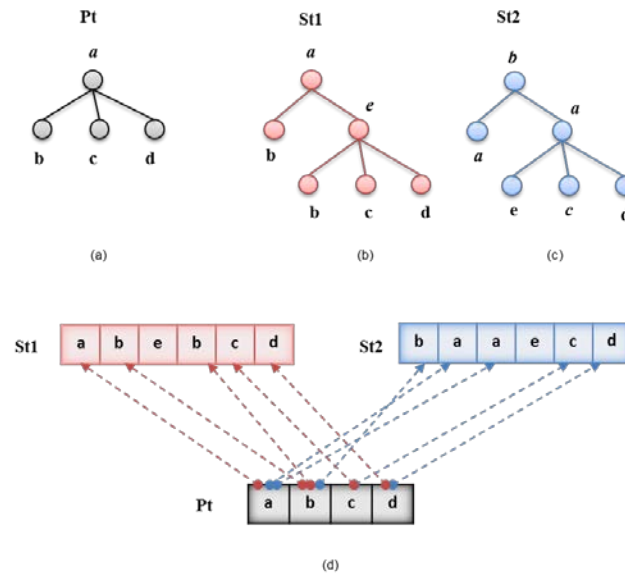


Figure 30: Node Mapping

Results of mappings are kept in a matrix called Node Mapping Matrix (NMM) which consists of linked lists with width equal to the size of Pt i.e. number of nodes, and height is unknown. The reason of using linked lists instead of using 2-dimensional matrices is that linked lists allow dynamic extension of the list size depending on the number of matching nodes. Figure 31 shows the NMM for the example in figure 30.

a	b	c	d
St1[0]	St1[1]	St1[4]	St1[5]
St2[2]	St1[3]	St2[4]	St2[5]
St2[1]	St2[0]		

Figure 31: The NMM for mappings in figure 30



### 5.4.2. Arc Mapping Matrix (AMM)

AMM's are created in the same way as of NMM's. First, arcs of a Pt are extracted and encoded using a 2-dimensional matrix having the first row for parent nodes and the second row for child nodes. Arc mapping can be thought of as pairs of node mappings in which two node mappings are performed for each arc. However, nodes that are mapped in NMM will not necessarily be included in AMM. For example, in figure32, the Pt arc (a, c) is mapped to the nodes a (St2[2]) and c (St2[4]) but NOT c(St2[1]). That is because the latter does not form an arc with the node a (St2[2]).

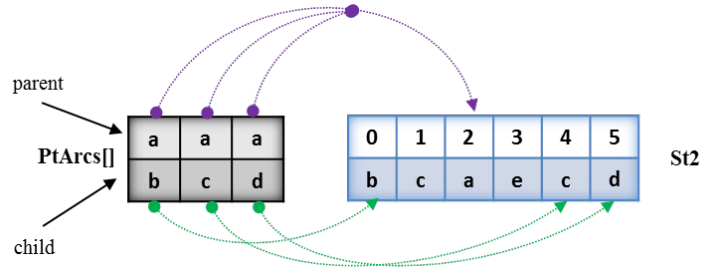


Figure 32: Arc Mapping

Arc mapping between a pattern tree Pt and a schema tree St is defined as a function  $\phi_A(Pt, St)$  such that:

$$\phi_A(Pt, St) = \{[(np, nc), (m, k), \rho] \mid (np, nc) \in A(Pt) \text{ and } m, k \in N(St)\}.$$

Where:

- $np$  is the parent node
- $nc$  is the child node
- $\rho$  is the arc matching type and  $\rho \in \{D, I, A, N, S, IS, SA, IN, NS, ISN\}$

Where D, I, A, N, S, IS, SA, IN, NS and ISN refer to Direct, Inverted, AttNode, Normalised, SepNode, InvertedSepNode, SepNodeAttNode, InvertedNormalised, NormalisedSepNode and InvertedSepNodeNormalised arc matching, respectively.

Therefore, AMM is a two dimensional matrix where the first row represents the pattern arcs and the following rows represent the matched schema arcs. For the example in figure30, the AMM will look like the one in figure33. A matched schema arc is a triple  $\langle Stx[j], Stx[k], \rho \rangle$  where  $Stx[j]$  and  $Stx[k]$  are nodes in the schema  $Stx$  and  $\rho$  is the type of arc match.

PtArcs[0]= A(a, b)	PtArcs[1]= A(a, c)	PtArcs[2]= A(a, d)
$\langle St1[0], St1[1], D \rangle$	$\langle St1[0], St1[4], S \rangle$	$\langle St1[0], St1[5], S \rangle$
$\langle St1[0], St1[3], S \rangle$	$\langle St2[2], St2[4], D \rangle$	$\langle St2[2], St2[5], D \rangle$
$\langle St2[0], St2[2], I \rangle$		

Figure 33: AMM for mappings in figure 18

#### 5.4.3. Query Index Matrix (QIM)

Since each pattern arc can be matched to more than one schema arcs, it is possible to have more than one answer to a certain query. Referring to the

AMM in figure33, there are three pattern arcs: PtArcs[0] which has three matching schema arcs, PtArcs[1] which has two matching arcs and PtArcs[2] which also has two matching arcs. Therefore, the final number of output queries will be equivalent to the total number of combinations of the matching schema arcs. i.e.

$$\begin{aligned} \text{Total number of output queries} &= \# \text{ of matching arcs of PtArcs[0]} \times \# \text{ of} \\ &\quad \text{matching arcs of PtArcs[1]} \times \# \text{ of matching arcs of PtArcs[2]} \\ &= 3 \times 2 \times 2 = 12 \text{ queries} \end{aligned}$$

**Definition 22: Number of output queries**

For a given pattern tree Pt , the number of output queries is given by:

$$\# \text{ of output queries} = |M(\text{PtArcs}[0])| \times |M(\text{PtArcs}[1])| \times \dots \times |M(\text{PtArcs}[i])|$$

Where:

i= number of arcs in Pt,  $M()$  is a function that return the matching schema arcs and  $|M()|$  is the number of matching schema arcs.

An index matrix is created to keep an index of matching arcs for each new query i.e. arcs that are used to construct each new query. The width of the matrix will be the size of PtArcs[] i.e. number of pattern arcs, and the height will be the number of output queries. For the previous example, and using

the AMM in figure34 (b) as input, the Query Index Matrix (QIM) will look like the one shown in figure34 (a).

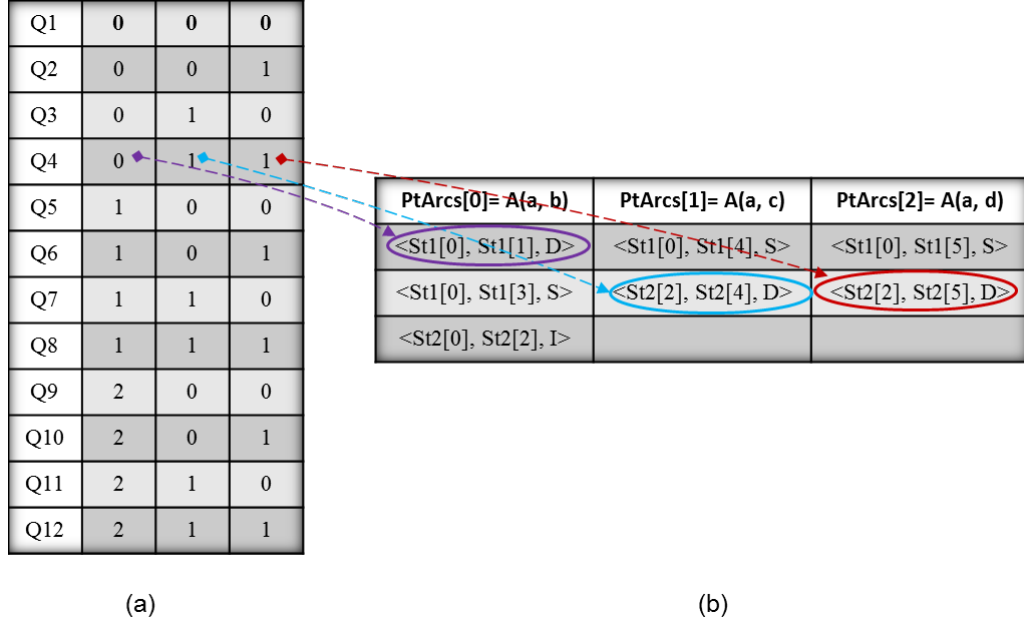


Figure 34: (a) QIM, (b) AMM

Thus, elements of QIM refer to elements of AMM. For example, Q4 is based on elements {0, 1, 1} which means that the new query Q4 will be constructed by joining the arcs <St1[0], St1[1], D>, <St2[2], St2[4], D> and <St2[2], St2[5], D>. It is assumed here that only full queries are considered i.e. queries that have matching schema arcs for all pattern arcs.

An algorithm is developed to generate indexes of new queries in QIM (figure35) and to filter out any intermediate (useless) queries. Eliminating intermediate queries is essential to obtain good performance, especially in case of big Pt's where the number of output queries can reach thousands.

### Generate New Queries Algorithm

Input: AMM

Output: Filtered QIM

Begin

For each combination of mapping arcs in AMM

    addQueryIndex() // insert index of participating arcs into QIM

For each record in QIM {

    If the query references one arc only of any St // not useful

        Delete query; // delete the record

    If the query reference two or more arcs of any St and non has an ID  
node

        Delete query; // delete the record

End

Figure 35: Generate New Queries Algorithm

The query filtration process described in the algorithm above passes by several stages to identify undesired queries. These stages are:

- **Stage 1:** Delete any query where one arc only of any schema tree is mapped. If the only referenced arc is an ID arc, then it is not useful because it needs to be joined with other twigs based on that ID arc which means that the same arc exists in other twigs of St's e.g. the arc  $A(dept, dname)$  of St1 in figure36. On the other hand, if it is not an ID arc e.g.  $A(dept, location)$  of St2 in the same figure, then we cannot join it with other twigs unless there is an ID arc in the same twig such as  $A(dept, dname)$ .
- **Stage 2:** Even if there is more than one matching arc in the same twig and none is an ID arc then it will not be useful as it cannot be

joined with other twigs e.g. St2 in figure36 has the group and location of departments but it does not have a *dname* node as an ID node for twig joins i.e. it is not possible to identify to which department the *location* and *group* nodes should be assigned.

- Stage 3:** Up to this point, more than 90% of the queries in the QIM are filtered out. Even though the remaining queries are meaningful, most of them are repetitive. For example, the resultant QIM of the case in figure 36 will have, among the new queries, two queries; one with A(dept, dname) from St3 and the other with the same arc from St4. Since that is an ID arc, it is required to be in both twigs if they are to be joined together; and the same output is obtained whether that arc is returned from St3 or St4. Therefore, one of these two queries is kept and the other is deleted. As a general rule, for any two queries in QIM, if the ID arcs composing the two queries are different and the non-ID arcs are similar, then the two queries are equivalent and one of them should be deleted.

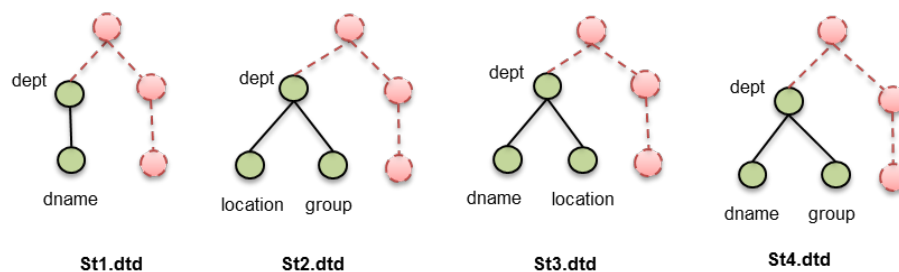


Figure 36: Different matching twigs

By eliminating these intermediate queries, the number of remaining queries will be much less and therefore more manageable.

## **5.5. Chapter conclusion**

To conclude, different techniques were presented in this chapter to soften the rules of matching a pattern tree  $P_t$  against a set of schema trees  $St$ 's on the element level (node matching) as well as on the structure level (arc matching). To the best of our knowledge, the proposed techniques of soft arc matching are novel and they are able to address the problem of querying and integrating heterogeneous data sources, particularly the structure of the data. A number of matrices were developed to process the node and arc mappings and the results were passed to an algorithm to rewrite the original query into a new one, which is the subject of the next chapter.

## **Chapter Six: XML Query Rewriting**

### **6. XML Query Rewriting**

6.1. Composing Queries

6.2. Query Rewriting Algorithms

6.3. Query Ranking

6.4. Chapter conclusion



## 6. XML Query Rewriting

The output of mapping pattern nodes and arcs with schema trees (DTDs) is stored in several matrices. Those are passed to an algorithm that rewrites the original query into a new query(s) according to the mappings. In this chapter we discuss our approach of joining softly matched schema arcs and twigs to construct answers to a pattern query. Additionally, we present novel algorithms for rewriting pattern queries based on mappings.

### 6.1. Composing Queries

Mappings provide information on where an answer of a Pt can be found. To compose an answer query, partial results from different sources need to be merged (joined) together. In other words, the matching schema arcs in the AMM (see previous chapter) need to be joined together in order to construct an answer (witness) tree. Before discussing the process of joining matching twigs, we present few definitions.

#### **Definition 23: Parent arcs and Child arcs**

An arc  $A(m, n)$  is said to be the parent of an arc  $A(k, l)$  if and only if the two arcs are in the same tree and  $n=k$  i.e. if the child node of the parent arc is the parent node of the child arc.

**Definition 24: Ancestor arcs and Descendant arcs**

An arc  $A(m, n)$  is said to be an ancestor of an arc  $A(k, l)$  if and only if the two arcs are in the same tree and  $n \in \text{anc}(k)$  i.e. if the child node of the parent arc is an ancestor (or parent) of the parent node of the child arc.

**Definition 25: Sibling arcs**

Two arcs  $A(m, n)$  and  $A(k, l)$  are said to be siblings if and only if they are in the same tree and  $m=k$  i.e. the two arcs have the same parent node.

Figure 37 shows examples on the aforementioned three definitions.

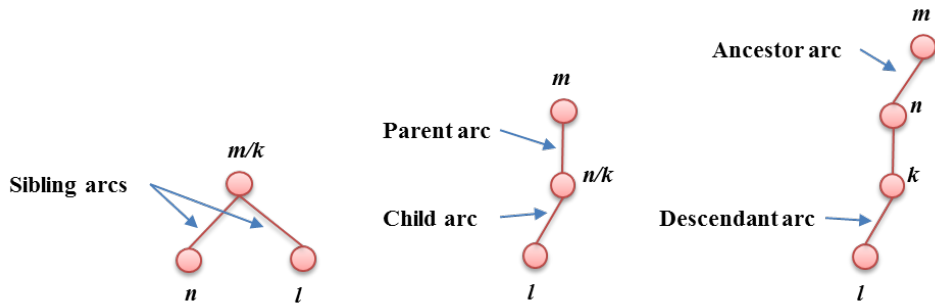


Figure 37: Examples on sibling, parent, child, ancestor and descendant arcs

For a pattern tree  $P_t$  with size  $x$  nodes, i.e.  $(x-1)$  arcs, a maximum of  $(x-2)$  arc joins are required in order to construct a witness tree. However, it is often the case where more than one pattern arc are matched against the same schema i.e. a pattern twig is matched. Figure 26 shows an example of matching twigs. The number of required joins equals the number of matching twigs-1 i.e. three joins.

Two or more matching schema arcs form a matching twig if they are sibling arcs, parent-child arcs, ancestor-descendant arcs or any combination of these relationships. Figure 38 shows examples of matched twigs in schema trees St2 and St3. In some cases, the matching schema arcs cannot form a single twig; they can result in more than one. In this case, the matching twigs are treated as if they belong to different schemas and they are joined together internally e.g. the schema St1 in figure 38.

Matching twigs are joined together via a common node that we call a Join node (see Definition 14). Generally, join nodes are found by anchoring the root of a matching twig into a node in another twig. However, if the root of a twig is part of an arc that has been matched using inverted arc matching, e.g.  $A(project, group)$  in St2 (figure 26), anchoring will be based on the child of that arc. In case of a normalized arc matching, a join within the schema is performed to de-normalized the matching twig i.e. to revert the ID/IDREF connection. Figure 26 shows how twigs are joined.

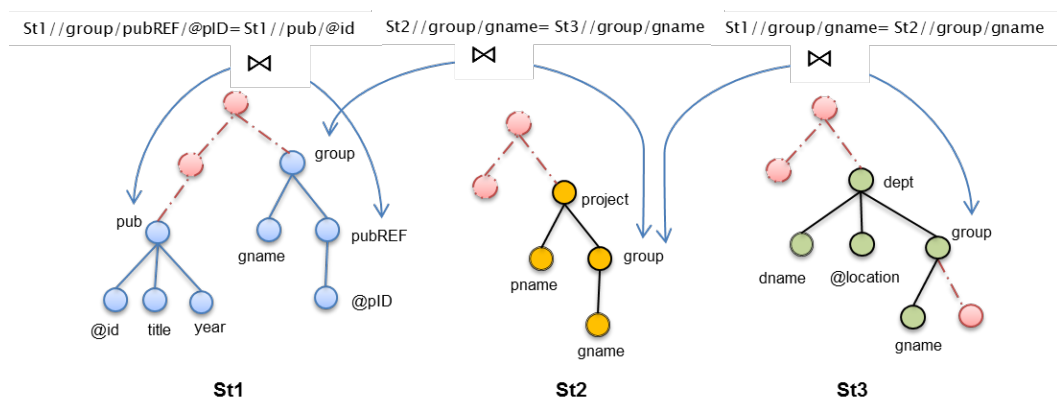


Figure 38: Joining matching twigs

**Definition 26: Twig Join**

If  $P_t$  is partially matched against a set of twigs  $T = T_1, \dots, T_n$ , then each twig  $T_i$  has to be joined with at least one twig  $T_j$  via a *join node*  $jNode$  such that:

$$T_i // jNode / [ID\ Node] = T_j // jNode / [ID\ Node]$$

Where  $jNode \in N(T_i)$  and  $jNode \in N(T_j)$  and  $jNode$  has a child ID node.

**Definition 27: Witness Tree**

Joining a set of Twigs  $T$  results in an answer to the original query, called *Witness Tree* ( $W_t$ ) given by:

$$W_t = T_1 \bowtie T_2 \bowtie \dots \bowtie T_n$$

The result of our matching approach is a set of mappings, node mappings and arc mappings, in addition to the matrix QIM. These are all passed to the next stage, query rewriting, where the original query is rewritten into a new one (or more) that is able to retrieve data from the matching data sources.

**6.2. Query Rewriting Algorithms**

Pattern queries are designed blindly based on a virtual target schema i.e. without being aware of the structure of underlying data sources. As such, and in order to return data from different local XML data sources, pattern queries need to be rewritten. For this study, we use an XQuery-like notation that

imitates the FLWOR expressions (see section 2.4.), the most common XQuery expression syntax. Figure39 shows a snippet of an XQuery that returns department name, group name, project name and publications of research groups from two data sources s1.xml and s2.xml.

```
for $d in doc("s1.xml")/dept, $g in doc("s2.xml")//group
where $d/group/gname=$g/gname and $d/location="London"
return
<group>
<departmentName>{data($d/dname)}
</departmentName>
{$g/gname}
{$d/dept/group/project/pname}
{$g/publication}
</group>
```

Figure39: An XQuery example

### Definition 28: XML Query

Inspired by the definition presented in [63], an XML query is a query of the following form:

$$\begin{aligned} & \text{for } G = \{x_1 \text{ in } g_1, x_2 \text{ in } g_2, \dots, x_n \text{ in } g_n\} \\ & \text{where } C = \{x_i/[ID] = x_j/[ID]\} \text{ where } 1 \leq i, j \leq n \\ & \text{return } R = \{x_i/[p]\} \end{aligned}$$

Where:

- G is a set of Generators; and a generator is a pair written as  $(\$x_n \text{ in } g_n)$  where  $\$x_n$  is a variable assigned to the XPath expression  $g_n$ . The latter usually represents the root of a matching schema twig.
- C is a set of Constraints  $C_1, C_2, \dots, C_k$  where  $C_i$  can be either a Join constraint of the form  $(\$x_i/[ID]=\$x_j/[ID])$ , Or a filter constraint of the form  $(\$x_i/[path] \text{ “operand” } [value])$
- O is a set of Outputs returned by the query, with each result has the form  $(\$x/[path])$

An algorithm is developed to translate mappings of pattern tree queries with local data sources into new queries that will be able to get data from these sources. The input of the algorithm is a set of matrices including i) pt Arcs: a matrix of pattern arcs and ii) AMM: a matrix of schema arcs matching correspondent pattern arcs and iii) QIM: an index matrix for new queries.

The algorithm shown in figure 40 derives the three main components of FLWOR expressions: Generators, Constraints and Outputs. Generators are the roots of matching twigs. Going back to the example in figure 26, generators will be the following:

- St1//group
- St1//pub
- St2//project

- St3//dept

As a query can be rewritten into more than one new query, generators are identified for each new query by investigating the correspondent arcs in the AMM. The algorithm starts by considering the parent of the first matching arc, say A1, as a generator and then iterates through the AMM. If the next arc, say A2, is not within the same schema tree (St), then the parent of A2 is added as a new generator. If A2 is in the same St, there are four possible cases:

- A1 and A2 are sibling arcs: in this case, no generators are added because the two arcs have the same parent node.
- A1 is an ancestor of A2: in this case, again no generators are added because the parent of A1 is an ancestor of the parent of A2.
- A2 is an ancestor of A1: in this case, the parent of A2 should be the generator and not the parent of A1; therefore, the parent of A1 is deleted from the generators and the parent of A2 is added.
- None of the above: in this case another generator with the parent of A2 is added, which means that there are two generators that belong to the same schema tree e.g. St1//group and St1//pub in figure 26.

## Query Rewriting Algorithm

Inputs: A matrix of Pattern tree arcs  $ptArcs[]$ , a matrix of arc mappings  $AMM[]$ , and an index matrix of new queries  $QIM[]$ .

Output: A set of Generators (G), Constraints (C) and Outputs (O).

Begin

generators[] = an array of linked lists of generators.

conditions[] = an array of linked lists of conditions.

outputs[] = an array of linked lists of outputs.

// derive generators

For each new query  $Q_i$  {

Generators.add(arcMatchList[i].get (0).parentNode) //add parent of the first arc

For each arc  $A_j$  in the arcMatchList[i]

For each generator  $g_k$  in G

    If (arc.parentNode) is an ancestor of  $g_i$  { // if the arc is ancestors of the generator node

        Generators.delete( $g_i$ ); //delete the old generator

        Generators.add (arc.parentNode); // insert the arc's parent node as a new generator

    }

// derive constraints

For each combination of generators ( $g_i, g_j$ ) {

    Search for a join node ( $n_j$ );

    If generators are related i.e. there is  $n_j$  {

        addConstraint( $g_i$  + "// $n_j$ " + getNodeID( $n_j$ ) = " $g_j$  + "// $n_j$ " + getNodeID( $n_j$ ));

    }

// derive outputs

For each node m in the patterNodes where node.returned='true' {

    For each generator in G

        If  $g_i$  is ancestor of m

        calculateRelativePath(m); // relative to the generator

        outputs.add(m.relativePath);

Figure 40: Query Rewriting Algorithm



It is not necessarily that there is only one generator per St. For example, in St1 (figure 38), there is a normalized match where we have two matching twigs one of them is referencing the other using ID/IDREF. In this case, two generators are created, one for each twig.

Constraints, particularly join constraints, specify how matching twigs are joined together. For that, the algorithm takes an array of generators, an arc mapping matrix AMM and Pt nodes information as input; and for each two generators, a constraint is formed based on several factors. First, depending on the type of arc match (Direct, Inverted, AttNode, Normalized, SepNode or Hybrid), the constraint is generated differently. For each couple of twigs, a “join node” is identified, and it is not necessarily the same node defined by the generator (i.e. the root of the matching twig), it might be a child of the generator node in case of inverted match e.g. St2 in figure 38. Additionally, in case of normalised arc match, e.g. St1 in figure 38, a constraint is formed to join internal twigs in order to de-normalise the ID/IDREF connections. Finally, the ID of each node involved in the join constraint is obtained in order to form an equi-join between join nodes of different twigs.

The last part is the set of Outputs. These are obtained by making use of the AMM, which links nodes and arcs of a Pt to their correspondents in St's. From Pt information, it can be verified whether a node is an output node or

not and then the relative path to that node is calculated by making use of the generators defined earlier.

### 6.3. Query Ranking

As mentioned in section 5.3.3, the Generate New Queries Algorithm (figure35) filters the queries in QIM in order to eliminate intermediate ones which removes around 90% of them. What remains (around 10%) are not all similar in terms of performance and confidence (Belief). Therefore, these queries can be ranked in order to meet users' requirements in terms of these two features.

Performance varies depending on the number of joins required for a certain query. By checking the remaining queries in QIM, a simple algorithm can determine the number of joins required for each one. For best performance, queries with minimum number of joins are selected.

For confidence, the arc matching degrees ( $\mu_A$ ) depends on the arc match type and therefore the total confidence of a query ( $\mu_{Qi}$ ) can be calculated as the total confidence for the arcs composing that query. More formally:

$$\mu_{Qi} = \sum_{k=0}^n \mu_{Ak}$$

Where the query  $Q_i$  is composed of the arcs  $A_0, A_1..A_k$ .

## 6.4. Chapter conclusion

In this chapter, the process of joining matching twigs was illustrated. In addition, novel algorithms were presented for XML pattern query rewriting. These use the outputs of the mapping phase as input in order to infer a new query(s). As XQuery is the standard language for XML queries, and because FLWOR is the most common XQuery expression, some FLWOR-like expressions were adopted as the format for both the original query and the new queries.

## **Chapter Seven: Experimental Results**

### **7. Experimental Results**

7.1. Prototype

7.2. Mapping phase

7.3. Query filtration and ranking

7.4. Query rewriting phase

7.5. Processing Cost

7.6. IFT vs. Other approaches

7.7. Chapter conclusion

## 7. Experimental Results

In this chapter, results of testing the proposed framework are presented. Testing was based on a set of synthetic data representing the research groups' information case study discussed in previous chapters. The main aspects covered by the testing are :i) ability of the proposed solution to identify all types of soft arc matching and to rewrite the original pattern query properly i.e. to provide the correct output ii) Processing cost i.e. CPU and memory usage and iii) scalability. Furthermore, testing was performed separately for the arc mapping phase and the query rewriting phase; and finally, for the two phases together.

### 7.1. Prototype

To test the proposed ideas, a prototype was developed with Java NetBeans IDE 6.8 on a 1.86 GHz PC with 3.0 GB of RAM. Pattern trees were modelled as XML documents which were parsed using the Java DOM Parser. Even though FLWOR expressions are not in XML format, an XML document representing a Pt can be extracted from there; however, this is beyond the scope of this study. Modelled as DTDs, schema trees were parsed using `org.xmlmiddleware.schemas.DTDs.*` package [78], which was developed with Java classes.

The testing sample consisted of 3 different Pts with different sizes, as shown in figure (41) below, and 50 DTDs, each of size 85 (nodes). Only 7 out of the 50 DTD had matching nodes with the Pts, the rest are random nodes from unrelated DTDs. This aims to test the scalability of the system by checking the cost (I/O and Memory) when matching a Pt against different sets of DTDs. Additionally, all types of soft arc matching addressed in section 5.2. are covered in the aforementioned DTDs.

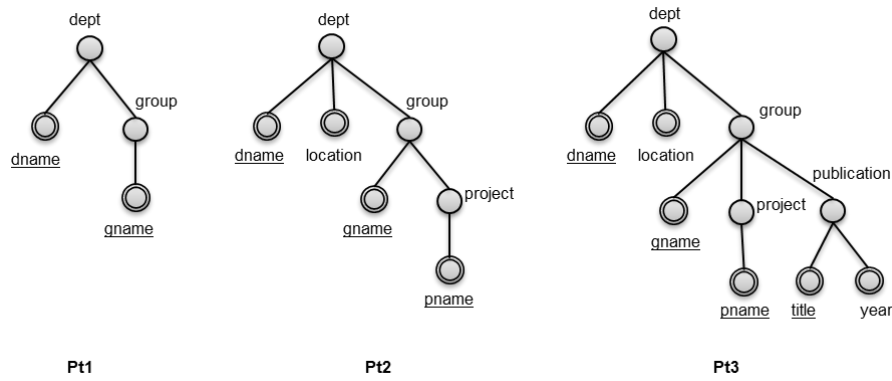


Figure 41: Pt's with different sizes

In the following sections, different features of the proposed solution are tested for the mapping phase, query writing phase and for the entire solution.

## 7.2. Mapping phase

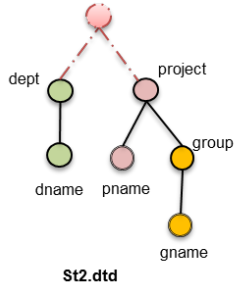
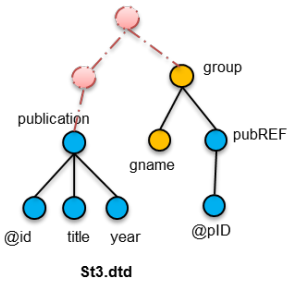
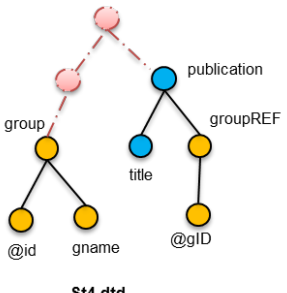
In this part, each Pt is compared against different sets of DTDs starting from 10 DTDs and adding 10 each time up to 50 DTDs. For each DTD, the fuzzy support ( $S_f$ ) and fuzzy confidence ( $C_f$ ) are calculated and soft node and arc

matching is performed. Results of matching (mappings) are kept in the correspondent mapping matrices which will be passed to next phase, query rewriting.

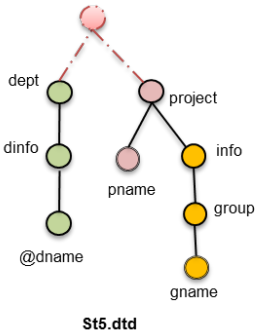
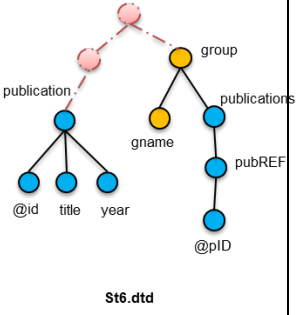
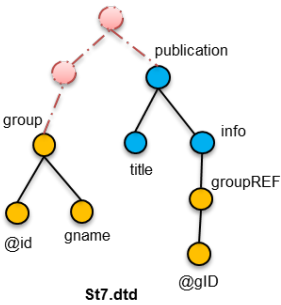
To start with, we show that all of the proposed types of soft arc matching were detected by the matching algorithm. We chose to match Pt3 against the 7 related DTDs so that all types of soft arc matching are tested. Each DTD is modelled as a schema tree and displayed alongside the matching arcs as in table1. In the left hand side of the table, each matching Pt arc is mapped to its counterpart in the correspondent St. The letters between the tags ‘<>’ indicate to the arc match type as discussed in section 5.2. A null tag, <null>, indicates that no matching is found.

Table(1): Results of node and arc mapping

Result from Java Netbeans IDE (Soft arc matching)	Schema Tree
<p>--- FuzzyMatch.matches()- c:/schemas/St1.dtd ----</p> <p><b>Pt[dept, dname]--&gt; St [dept, dname]&lt;D&gt;</b></p> <p><b>Pt[dept, location]--&gt; St [dept, @location]&lt;A&gt;</b></p> <p><b>Pt[dept, group]--&gt; St [dept, group]&lt;S&gt;</b></p> <p><b>Pt[group, gname]--&gt; St [group, gname]&lt;D&gt;</b></p>	<pre> graph TD     dept((dept)) --- dname((dname))     dept --- location((@location))     dept --- groups((groups))     groups --- group_star((group*))     group_star --- gname((gname))     </pre> <p>St1.dtd</p>

<p>--- FuzzyMatch.matches()- c:/schemas/St2.dtd ----</p> <p>Pt [dept, dname]--&gt; St [dept, @dname]&lt;A&gt;</p> <p>Pt [dept, group]--&gt; St [dept, group]&lt;null&gt;</p> <p>Pt [group, gname]--&gt; St [group, gname]&lt;D&gt;</p> <p>Pt [group, project]--&gt; St [group, project]&lt;I&gt;</p> <p>Pt [project, pname]--&gt; St [project, pname]&lt;D&gt;</p>	 <p>The diagram shows a schema tree for St2.dtd. The root is a red dashed circle. It has two children: 'dept' (green circle) and 'project' (pink circle). 'dept' has a child 'dname' (green circle). 'project' has two children: 'pname' (pink circle) and 'group' (yellow circle). 'group' has a child 'gname' (yellow circle). The label 'St2.dtd' is at the bottom.</p>
<p>--- FuzzyMatch.matches()- c:/schemas/St3.dtd ----</p> <p>Pt [group, gname] --&gt; St [group, gname]&lt;D&gt;</p> <p>Pt [group, publication] --&gt; St [group, publication]&lt;N&gt;</p> <p>Pt [publication, year] --&gt; St [publication, year]&lt;D&gt;</p> <p>Pt [publication, title] --&gt; St [publication, title]&lt;D&gt;</p>	 <p>The diagram shows a schema tree for St3.dtd. The root is a red dashed circle. It has two children: 'publication' (blue circle) and 'group' (yellow circle). 'publication' has three children: '@id' (blue circle), 'title' (blue circle), and 'year' (blue circle). 'group' has two children: 'gname' (yellow circle) and 'pubREF' (blue circle). 'pubREF' has a child '@pID' (blue circle). The label 'St3.dtd' is at the bottom.</p>
<p>--- FuzzyMatch.matches()- c:/schemas/St4.dtd ----</p> <p>Pt [group, gname]--&gt; St [group, gname]&lt;D&gt;</p> <p>Pt [group, publication]--&gt; St [group, publication]&lt;IN&gt;</p> <p>Pt [publication, title] --&gt; St [publication, title]&lt;D&gt;</p>	 <p>The diagram shows a schema tree for St4.dtd. The root is a red dashed circle. It has two children: 'group' (yellow circle) and 'publication' (blue circle). 'group' has two children: '@id' (yellow circle) and 'gname' (yellow circle). 'publication' has two children: 'title' (blue circle) and 'groupREF' (yellow circle). 'groupREF' has a child '@gID' (yellow circle). The label 'St4.dtd' is at the bottom.</p>



<p>--- FuzzyMatch.matches()- c:/schemas/St5.dtd ----</p> <p>Pt[dept, dname]--&gt; St [dept, @dname]&lt;AS&gt;</p> <p>Pt[dept, group]--&gt; St [dept, group]&lt;null&gt;</p> <p>Pt[group, gname]--&gt; St [group, gname]&lt;D&gt;</p> <p>Pt[group, project]--&gt; St [group, project]&lt;IS&gt;</p> <p>Pt[project, pname]--&gt; St [project, pname]&lt;D&gt;</p>	 <p>The diagram shows the schema tree for St5.dtd. The root is a red dashed circle. It has two children: 'dept' (green circle) and 'project' (pink circle). 'dept' has a child 'dinfo' (green circle), which has a child '@dname' (green circle). 'project' has two children: 'pname' (pink circle) and 'info' (yellow circle). 'info' has a child 'group' (yellow circle), which has a child 'gname' (yellow circle).</p>
<p>--- FuzzyMatch.matches()- c:/schemas/St6.dtd ----</p> <p>Pt [group, gname]--&gt; St [group, gname]&lt;D&gt;</p> <p>Pt[group, publication]--&gt; St [group, publication]&lt;SN&gt;</p> <p>Pt[publication, year]--&gt; St [publication, year]&lt;D&gt;</p> <p>Pt[publication, title]--&gt; St [publication, title]&lt;D&gt;</p>	 <p>The diagram shows the schema tree for St6.dtd. The root is a red dashed circle. It has two children: 'publication' (blue circle) and 'group' (yellow circle). 'publication' has three children: '@id' (blue circle), 'title' (blue circle), and 'year' (blue circle). 'group' has two children: 'gname' (yellow circle) and 'publications' (blue circle). 'publications' has a child 'pubREF' (blue circle), which has a child '@pID' (blue circle).</p>
<p>--- FuzzyMatch.matches()- c:/schemas/St7.dtd ----</p> <p>Pt[group, gname]--&gt; St [group, gname]&lt;D&gt;</p> <p>Pt[group, publication]--&gt; St [group, publication]&lt;ISN&gt;</p> <p>Pt[publication, title]--&gt; St [publication, title]&lt;D&gt;</p>	 <p>The diagram shows the schema tree for St7.dtd. The root is a red dashed circle. It has two children: 'group' (yellow circle) and 'publication' (blue circle). 'group' has two children: '@id' (yellow circle) and 'gname' (yellow circle). 'publication' has two children: 'title' (blue circle) and 'info' (blue circle). 'info' has a child 'groupREF' (yellow circle), which has a child '@gID' (yellow circle).</p>

### 7.3. Query filtration and ranking

The output of the mapping phase, particularly the AMM, is passed as input to the Generate New Queries Algorithm (figure 35) along with PtNodes matrix. The algorithm generates indexes of new queries (QIM), including intermediate ones and then filters out the latters.

Following the same testing case study presented in section 7.2., the total number of output queries in QIM will be the result of multiplying the number of matching schema arcs for each pattern arc in Pt as follows: (See Definition 22)

Number of output queries= $|M(A(\text{dept}, \text{dname}))| \times |M(A(\text{dept}, \text{location}))| \times |M(A(\text{dept}, \text{group}))| \times |M(A(\text{group}, \text{gname}))| \times |M(A(\text{group}, \text{project}))| \times |M(A(\text{group}, \text{publication}))| \times |M(A(\text{project}, \text{pname}))| \times |M(A(\text{publication}, \text{title}))| \times |M(A(\text{publication}, \text{year}))|$

$$= 3 \times 1 \times 1 \times 7 \times 2 \times 4 \times 2 \times 4 \times 2$$

$$= 2,688$$

After performing stage one of filtration (see section 5.3.3.), which deletes any query where one arc only of any schema tree is mapped, only 76 out of 2,688 queries remained. In other words, 94% of the queries were deleted.

The remaining 76 queries went in the second stage where queries with twigs that do not have an ID arc are excluded. This left only 64 for the next stage.

Now in the final stage, if the ID arcs composing any two queries are different and the non-ID arcs are similar, then the two queries are equivalent and one of them should be deleted. By doing that, only 12 queries remained.

Overall, the filtration algorithm kept 16 queries out of 2,688 i.e. almost 0.6% of the queries in QIM. Index of these queries is shown in figure 42.

	(dept, dname)	dept, location	(dept, group)	(group, gname)	(group, project)	(group, pub)	(project, pname)	(pub, year)	(pub, title)	# sources	$\mu$
Q[1]	[0]: s1.dtd	[0]: s1.dtd	[0]: s1.dtd	[0]: s1.dtd	[0]: s2.dtd	[0]: s3.dtd	[0]: s2.dtd	[0]: s3.dtd	[0]: s3.dtd	3	93%
Q[2]	[0]: s1.dtd	[0]: s1.dtd	[0]: s1.dtd	[0]: s1.dtd	[0]: s2.dtd	[2]: s6.dtd	[0]: s2.dtd	[1]: s6.dtd	[1]: s6.dtd	3	90%
Q[3]	[0]: s1.dtd	[0]: s1.dtd	[0]: s1.dtd	[0]: s1.dtd	[1]: s5.dtd	[0]: s3.dtd	[1]: s5.dtd	[0]: s3.dtd	[0]: s3.dtd	3	90%
Q[4]	[0]: s1.dtd	[0]: s1.dtd	[0]: s1.dtd	[0]: s1.dtd	[1]: s5.dtd	[2]: s6.dtd	[1]: s5.dtd	[1]: s6.dtd	[1]: s6.dtd	3	87%
Q[5]	[0]: s1.dtd	[0]: s1.dtd	[0]: s1.dtd	[2]: s3.dtd	[0]: s2.dtd	[0]: s3.dtd	[0]: s2.dtd	[1]: s6.dtd	[1]: s6.dtd	4	93%
Q[6]	[0]: s1.dtd	[0]: s1.dtd	[0]: s1.dtd	[2]: s3.dtd	[0]: s2.dtd	[2]: s6.dtd	[0]: s2.dtd	[0]: s3.dtd	[1]: s6.dtd	4	90%
Q[7]	[0]: s1.dtd	[0]: s1.dtd	[0]: s1.dtd	[2]: s3.dtd	[1]: s5.dtd	[0]: s3.dtd	[1]: s5.dtd	[1]: s6.dtd	[1]: s6.dtd	4	90%
Q[8]	[0]: s1.dtd	[0]: s1.dtd	[0]: s1.dtd	[2]: s3.dtd	[1]: s5.dtd	[2]: s6.dtd	[1]: s5.dtd	[0]: s3.dtd	[1]: s6.dtd	4	87%
Q[9]	[0]: s1.dtd	[0]: s1.dtd	[0]: s1.dtd	[3]: s4.dtd	[0]: s2.dtd	[1]: s4.dtd	[0]: s2.dtd	[0]: s3.dtd	[0]: s3.dtd	4	90%
Q[10]	[0]: s1.dtd	[0]: s1.dtd	[0]: s1.dtd	[3]: s4.dtd	[0]: s2.dtd	[1]: s4.dtd	[0]: s2.dtd	[1]: s6.dtd	[1]: s6.dtd	4	90%
Q[11]	[0]: s1.dtd	[0]: s1.dtd	[0]: s1.dtd	[3]: s4.dtd	[1]: s5.dtd	[1]: s4.dtd	[1]: s5.dtd	[0]: s3.dtd	[0]: s3.dtd	4	87%
Q[12]	[0]: s1.dtd	[0]: s1.dtd	[0]: s1.dtd	[3]: s4.dtd	[1]: s5.dtd	[1]: s4.dtd	[1]: s5.dtd	[1]: s6.dtd	[1]: s6.dtd	4	87%
Q[13]	[0]: s1.dtd	[0]: s1.dtd	[0]: s1.dtd	[6]: s7.dtd	[0]: s2.dtd	[3]: s7.dtd	[0]: s2.dtd	[0]: s3.dtd	[0]: s3.dtd	4	89%
Q[14]	[0]: s1.dtd	[0]: s1.dtd	[0]: s1.dtd	[6]: s7.dtd	[0]: s2.dtd	[3]: s7.dtd	[0]: s2.dtd	[1]: s6.dtd	[1]: s6.dtd	4	89%
Q[15]	[0]: s1.dtd	[0]: s1.dtd	[0]: s1.dtd	[6]: s7.dtd	[1]: s5.dtd	[3]: s7.dtd	[1]: s5.dtd	[0]: s3.dtd	[0]: s3.dtd	4	86%
Q[16]	[0]: s1.dtd	[0]: s1.dtd	[0]: s1.dtd	[6]: s7.dtd	[1]: s5.dtd	[3]: s7.dtd	[1]: s5.dtd	[1]: s6.dtd	[1]: s6.dtd	4	86%

Figure 42: Remaining queries in QIM

As shown in the figure, the top row consists of the pattern arcs and the rows below contain an index of the schema arc that maps to the pattern arc between brackets '['']' as well as the schema name e.g. s1.dtd. The column titled "# sources" indicated to how many schemas the query is composed from. Finally the column  $\mu$  indicates to the degree of Belief or matching degree of that query, which is the total of beliefs for all composing arcs divided by the number of arcs.

As mentioned before, the proposed solution allows users to choose between high performance and confidence. For best performance, queries Q[1]-Q[4] are the best choices as the number of required joins will be two joins only whereas for the other queries 3 joins are required. Number of required joins is simply the number of data sources-1. Even though the difference between the above queries is one join only, it does make a difference when dealing with big data sources.

In case users' priority is precision rather than performance, they can choose queries with high  $\mu$ . For the queries in the previous figure, Q1 and Q5 have belief of 93% which make them the most trusted queries.

#### **7.4. Query rewriting phase**

The remaining queries in QIM are passed to the following stage, query rewriting. In fact, what is left in QIM are just mappings of pattern arcs with

correspondent schema arcs. Only the first seven DTDs had matching arcs whereas the rest are dummy DTDs for the purpose of testing the scalability of the system. After running the prototype on the first 10 DTDs and filtering and rewriting the queries in QIM, the final output i.e. the new queries are produced. For space limitations, we just show the result of rewriting the first query (Q1) in figure 43 below.

```

----- XQueryRewrite.printNewQueries() -----
Query [1]
Generators...
$x1 in doc('c://schemas//all//s1.dtd')/site/university/dept
$x2 in doc('c://schemas//all//s2.dtd')/site/university/project
$x3 in doc('c://schemas//all//s3.dtd')/site/group
$x4 in doc('c://schemas//all//s3.dtd')/site/publication
-----
Constraints...
$x1//group/gname=$x3//gname
$x1//group/gname=$x2//group/gname
$x3//pubREF/@pID=$x4//@id
-----
Outputs...
$x1/dname
$x1/@location
$x3/gname
$x2/pname
$x4/title
$x4/year

```

Figure 43: A new query with main components of FLWOR expression

The new query, shown in the previous figure, is not shown as a FLWOR expression. It is shown as a set of Generators, Filters and Outputs which are the main parts required for constructing a FLWOR expression.

## 7.5. Processing Cost

It is of great importance to test how much resources i.e. CPU and memory are consumed by the proposed solution. For that purpose, NetBeans Profiler was utilized. Pt3in figure 41 was compared against 10, 20, 30, 40 and 50 DTDs (each is of 85 nodes size); and performance was analysed in terms of CPU time (seconds) and memory usage (Mega Bytes). Additionally, processing cost was analysed for individual tasks such as query matching and query rewriting as well as for the entire solution. Figure 44 and 45 show line charts for performance and memory consumption respectively.

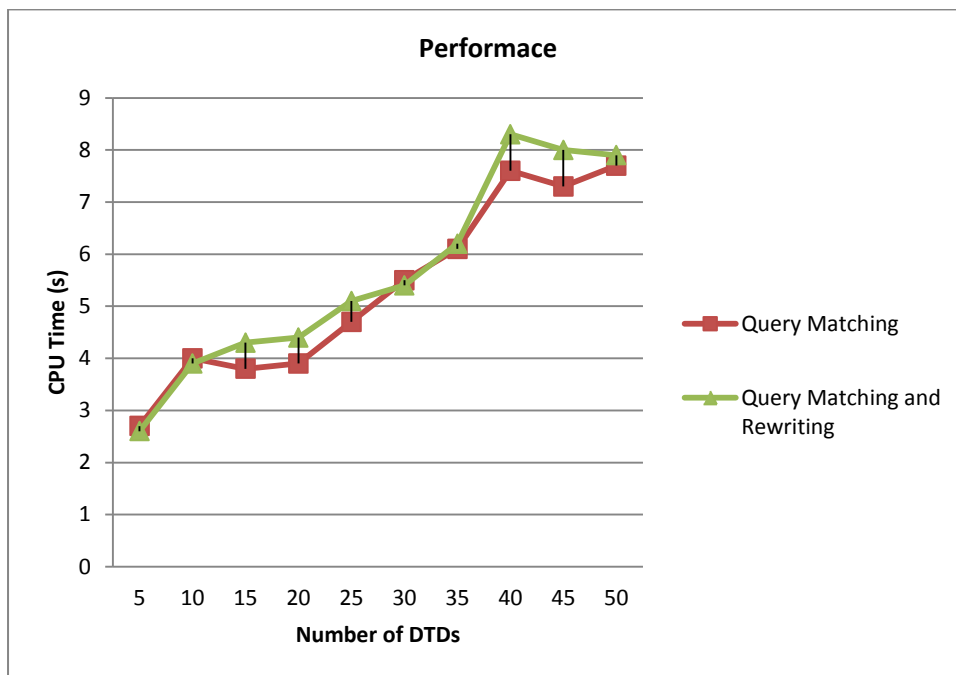


Figure 44: Performance results

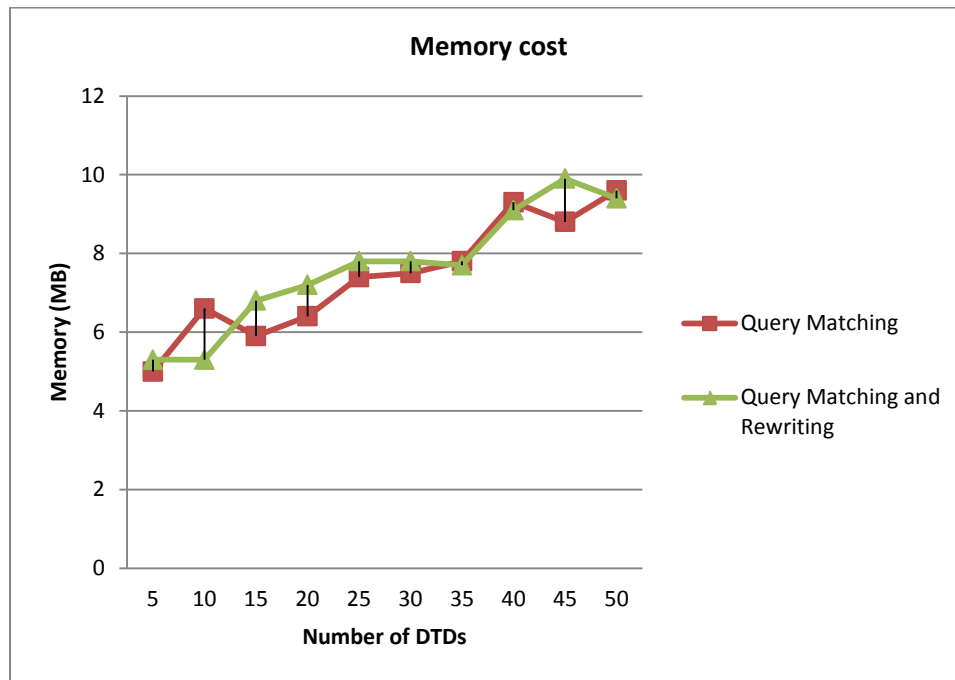


Figure 45: Memory consumption results

The performance figure shows that the CPU time increases almost linearly as the number of DTDs increases for both query matching and query rewriting tasks. Most of the processing time is spent on the query matching part. This can be noticed clearly as the curves of query matching and the curve of query matching and writing are very close.

The memory cost figure, on the other hand, shows that memory consumption increases reasonable with the increment in number of DTDs. Starting with around 5.0 MB for 5 DTDs, memory consumption is doubled (i.e. 10 MB) when the DTDs are increase by 10 times (50 DTDs) which indicates that the system is highly scalable. The query matching (Red curve) and query



matching and rewriting (Green curve) curves are very close; they are actually intersecting each other at some points. For the 10 DTDs point however, the green curve goes below the red one, which is a testing anomaly caused by the Java profiler. Overall, the figure implies that query rewriting does not require any extra memory and that the query matching consumes most of the heap memory allocated to the system.

The aforementioned memory cost refers to the maximum used heap at any time during the prototype execution; the average memory usage is always less than that. Figure 46 and 47 below show the amount of used heap memory at each moment of the execution for 10 DTDs and 50 DTDs respectively.

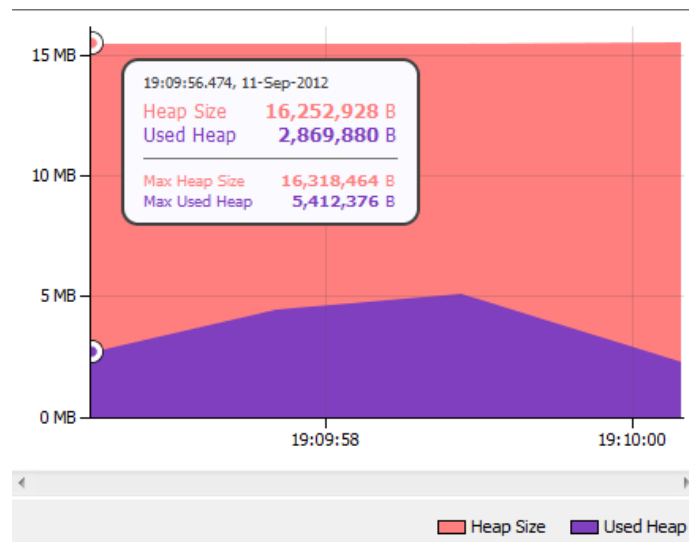


Figure 46: Heap memory usage during execution time for 10 DTDs

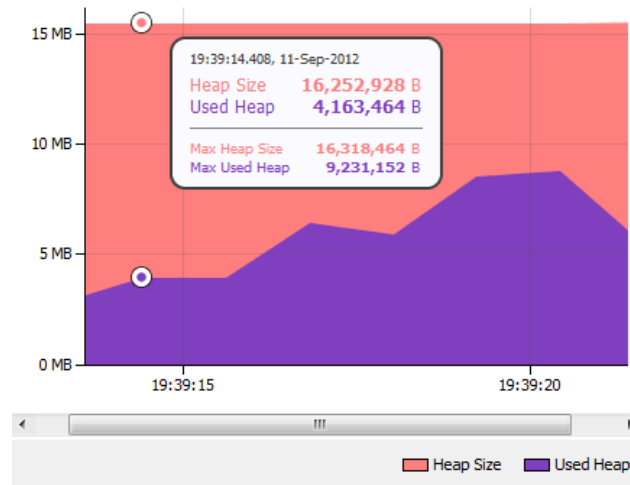


Figure 47: Heap memory usage during execution time for 50 DTDs

## 7.6. IFT vs. Other approaches

IFT has been compared against previous approaches according to a number of features as shown in table (2). First, approaches were compared against different types of arc matching. Obviously, they all support direct arc matching while few of them support other types such as Normalised and SepNode matching. However, the Inverted, AttNode and Hybrid match types are only supported by IFT and cannot be identified using any other approach.

Next, the approaches were compared against a set of features. Those are:

- Partial results: an indicator to whether the approach can obtain partial results (matchings) from multiple data sources and then join these sub-results to construct a full answer to a query.

Table (2): IFT vs. Other approaches

	Arc match type						Other features				
Approach/ Feature	Direct	Inverted	Normalised	SepNode	AttNode	Hybrid	Partial results	Ranking	Logical Operators	Purpose /Style	Content/Schema match
TED [28-30]	Yes	No	No	Yes	No	No	No	No	No	Similarity	[28-29] schema [30] Both
Twig Patterns [31-38, 40, 51]	Yes	No	No	No	No	No	Yes	No	No	Evaluation	Both
Query Relaxation [14-20]	Yes	No	No	Yes	No	No	No	Yes	No	IR	Schema
Fuzzy PTs [41, 42]	Yes	No	No	No	No	No	No	No	No	Matching	Both
Extended PTs [52-55]	Yes	No	No	No	No	No	No	No	Yes	Matching	Schema
Graph Patterns [57-58]	Yes	No	Yes	No	No	No	Yes	No	No	Matching	Schema
Tree Algebra [61]	Yes	No	No	No	No	No	No	No	No	Matching	Schema
Tree Mining [12,13, 62]	Yes	No	No	No	No	No	No	No	No	Similarity	Schema
Query Rewriting [63, 64]	Yes	No	Yes	Yes	No	No	Yes	Yes	No	Rewriting	Schema
IFT matching	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Similarity, Matching & Rewriting	Schema

- Ranking: refers to whether the approach contains ranking functionality for the query results.
- Logical operators: does the approach support logical operators such as OR, AND and negation function.
- Purpose/style: it can be either Tree similarity, query evaluation, query matching, query rewriting or IR(Information Retrieval)
- Content/schema match: refers to whether the scope is to match content, schema or both.

As shown, IFT supports main features such as partial results and ranking and covers many purposes such as similarity, matching and rewriting.

## **7.7.Chapter conclusion**

The chapter described the prototype developed to verify the proposed solution. Different aspects were investigated by testing individual parts of the prototype and then testing the whole system all together. Results showed that the system was able to identify different types of soft arc matching, assign appropriate belief/matching degree to each one of them and use these matchings to generate new queries. Additionally, it was proved that the proposed algorithms do not consume a lot of resources even with big numbers of data sources.

## **Chapter Eight: Conclusion and Further Work**

### **8. Conclusion and Further Work**

#### **8.1. Summary**

#### **8.2. Contributions and limitations**

#### **8.3. Directions for further work**

## **8. Conclusion and Further Work**

In this chapter, a summary of this thesis is presented. In addition, the contributions and limitations of this research are pointed out. Directions for further research are also addressed in the last section.

### **8.1. Summary**

This thesis addressed the issue of approximate matching and rewriting of XML queries using IFT. It started by introducing the XML data model and its features and critically evaluated the benefits and drawbacks of that model. The most popular XML query languages were discussed briefly with main focus on XQuery as a W3C recommendation.

A comprehensive literature review of XML similarity and pattern tree matching was presented. Traditional schema matching approaches were discussed as well as XML schema matching (or similarity) approaches. XML query matching approaches such as Tree Edit Distance, Pattern Tree Matching and others were thoroughly evaluated and classified. Great attention was given to studies on structural pattern tree matching and their limitations were pointed out. Additionally, relevant XML query rewriting approaches were investigated. Overall, the previous studies revealed limited

ability of efficiently querying XML documents from different data sources with different schemas.

After that, Intuitionistic Fuzzy Logic and IFT were presented. A set of definitions and formal equations on fuzzy tree similarity/inclusion were introduced. A new similarity measure based on Fuzzy Support and Fuzzy Confidence was demonstrated along with a novel algorithm for calculating it. Moreover, a novel approach for soft node matching and soft arc matching was demonstrated along with formalism for all different types of soft matching and matrices for holding results of matching. Furthermore, original algorithms were developed to use the results of soft arc matching in order to rewrite the original pattern query into new queries that can return data from available data sources even if they have different structures. More interestingly, the proposed algorithms were developed to obtain partial results from different sources and merge/join these results together in order to return a unified answer to the pattern query.

A prototype of the proposed solution was implemented and tested using Java NetBeans API. All different aspects of the solution such as the ability of matching pattern queries with heterogeneous XML documents and rewriting the original query in the light of matching results, were tested. In addition, performance and memory consumptions were tested for different sets of

DTDs and it proved that the proposed solution performs very well and does not consume a lot of resources even in case of big number of XML schemas.

## 8.2. Contributions and limitations

The proposed solution presented a new approach to approximate XML query matching and rewriting which proved to be more efficient than previous ones. This was achieved by a number of novel algorithms for soft matching of XML pattern trees and schema trees. Overall, the author claims the following contributions:

- a) Presenting IFT, a new approach for fuzzy tree similarity/inclusion based on considering the number of common nodes as well as the number of common arcs as basic units of data schema (structure) along with a two-value measure  $\langle Cf, Sf \rangle$  that indicates to what degree a tree is included in another.
- b) Introducing new types of fuzzy arc matching that can match a pattern arc to a schema arc as long as the correspondent parent and child nodes are there and have reachability between each other.
- c) Defining membership degrees for different types of soft arc matching and use these to calculate the degree of confidence of the composing query and rank new queries according to that.



- d) Proposing a novel algorithm to join matching arcs from different XML schemas based on ID constraints and uses these to construct new queries. Additionally, the algorithm consists of filtering new queries by removing intermediate ones.
- e) Producing a novel algorithm that takes arc mapping matrices as input and produces new queries in XQuery format, particularly in FLWOR expressions.

To the best of the author's knowledge, the above contributions are novel and no other previous studies proposed any similar contribution.

In regards to limitations, the proposed approach was limited to XML query (or schema) matching because both XML queries and XML documents can be modelled as trees and the approach applies to matching models that have tree structures including comparing XML schemas together for integration and clustering purposes.

Moreover, the proposed approach cannot handle all sorts of structural heterogeneity. In cases where one node such as "name" is modelled as two nodes "first name" and "last name", this cannot be resolved by our approach.

Furthermore, the study is focused on matching the structures of XML queries and documents; matching contents is not in the scope of this research nor query processing and optimisation are.

### 8.3. Directions for further work

In this thesis, a number of contributions have been achieved towards solving the issue of querying XML data sources with heterogeneous schemas. However, there is still a lot of work to be done before the issue is resolved efficiently. This can be summarised by the following:

- Integrate or query XML documents with different schemas where data and meta-data are mixed e.g. the tag <Africa> refers to a name of area i.e. it is data; however, it is treated as meta-data in this example.
- Develop query rewriting algorithms for different types of XML query languages such as XPath and for P2P schema translation.
- Extend IFT query matching approach to match contents in addition to structures. This can be useful in identifying and removing duplications.
- There is potential to improve the current approach by using semantic web technologies such as using reasoning to improve soft arc matching approaches.

These are few suggestions for research directions that the author thinks are worthwhile investigating. The way to efficient XML query languages that

can come over the high diversity in data representation is still far, but it is essential that a great amount of effort is invested in this area especially that XML is becoming the backbone of online data sources.

## References

- [1] W3C. (2012, 20 June 2011). *Introduction to XML*. Available: [http://www.w3schools.com/xml/xml\\_what.asp](http://www.w3schools.com/xml/xml_what.asp)
- [2] W3C. (2010, 21 June 2011). *An XML Query Language (Second Edition)* Available: <http://www.w3.org/TR/xquery/#id-introduction>
- [3] E. Harold and W. Means, *XML in a Nutshell*: O'Reilly Media, Incorporated, 2004.
- [4] Microsoft. (2011, 20 June 2012). *Config.xml file in Office 2010* Available: <http://technet.microsoft.com/en-us/library/cc179195.aspx>
- [5] B. Evjen, K. Sharkey, T. Thangarathinam, M. Kay, A. Vernet, and S. Ferguson, *Professional XML*: Wiley, 2007.
- [6] G. Powell, *Beginning XML Databases*: Wiley, 2007.
- [7] F. Ravat, O. Teste, R. Tournier, and G. Zurfluh, "Finding an application-appropriate model for XML data warehouses," *Information Systems*, vol. 35, pp. 662-687, Sep 2010.
- [8] B. Bos. (2005, 16/04/2012). *The XML data model*. Available: <http://www.w3.org/XML/Datamodel.html>
- [9] B. K. Y. Sagiv, "Twig Patterns: From XML Trees to Graphs " presented at the Ninth International Workshop on the Web and Databases (WebDB 2006), Chicago, Illinois., 2006.
- [10] J. Cheng, J. X. Yu, B. Ding, P. S. Yu, and H. Wang, "Fast Graph Pattern Matching," presented at the Proceedings of the 2008 IEEE 24th International Conference on Data Engineering, 2008.
- [11] C. Jiefeng, J. X. Yu, and P. S. Yu, "Graph Pattern Matching: A Join/Semijoin Approach," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 23, pp. 1006-1021, 2011.
- [12] F. D. R. Lopez, A. Laurent, P. Poncelet, and M. Teisseire, "FTMnodes: Fuzzy tree mining based on partial inclusion," *Fuzzy Sets Syst.*, vol. 160, pp. 2224-2240, 2009.
- [13] A. Laurent, M. Teisseire, and P. Poncelet, "Chapter 12 Fuzzy data mining for the semantic web: Building XML mediator schemas," in

*Capturing Intelligence*. vol. Volume 1, S. Elie, Ed., ed: Elsevier, 2006, pp. 249-264.

- [14] S. Amer-Yahia, S. Cho, and D. Srivastava, "Tree Pattern Relaxation," presented at the Proceedings of the 8th International Conference on Extending Database Technology: Advances in Database Technology, 2002.
- [15] S. Amer-Yahia, L. V. S. Lakshmanan, and S. Pandit, "FlexPath: flexible structure and full-text querying for XML," presented at the Proceedings of the 2004 ACM SIGMOD international conference on Management of data, Paris, France, 2004.
- [16] C. Liu, J. Li, J. X. Yu, and R. Zhou, "Adaptive relaxation for querying heterogeneous XML data sources," *Inf. Syst.*, vol. 35, pp. 688-707, 2010.
- [17] N. Wiwatwattana, H. V. Jagadish, L. V. S. Lakshmanan, and D. Srivastava, "X<sup>3</sup>: A Cube Operator for XML OLAP," in *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, 2007, pp. 916-925.
- [18] B. Fazzinga, S. Flesca, and A. Pugliese, "Retrieving XML data from heterogeneous sources through vague querying," *ACM Trans. Internet Technol.*, vol. 9, pp. 1-35, 2009.
- [19] B. Fazzinga, S. Flesca, and F. Furfaro, "On the expressiveness of generalization rules for XPath query relaxation," presented at the Proceedings of the Fourteenth International Database Engineering & Applications Symposium, Montreal, Quebec, Canada, 2010.
- [20] B. Fazzinga, S. Flesca, and F. Furfaro, "XPath Query Relaxation through Rewriting Rules," *IEEE Trans. on Knowl. and Data Eng.*, vol. 23, pp. 1583-1600, 2011.
- [21] E. Rahm and P. A. Bernstein, "A survey of approaches to automatic schema matching," *The VLDB Journal*, vol. 10, pp. 334-350, 2001.
- [22] P. Shvaiko and J. Euzenat, "A Survey of Schema-Based Matching Approaches Journal on Data Semantics IV," in *Journal on Data Semantics IV*. vol. 3730, S. Spaccapietra, Ed., ed: Springer Berlin / Heidelberg, 2005, pp. 146-171.

- [23] J. Tekli, R. Chbeir, and K. Yetongnon, "An overview on XML similarity: Background, current trends and future directions," *Computer Science Review*, vol. 3, pp. 151-173, 2009.
- [24] G. Gang, "Efficiently Querying Large XML Data Repositories: A Survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 19, pp. 1381-1403, 2007.
- [25] M. M. Sukomal Pal, "XML Retrieval: A Survey " *Internet Policies and Issues*, vol. 8, pp. 229-272, 2006.
- [26] H. Marouane, "A Survey of XML Tree Patterns," *IEEE Transactions on Knowledge and Data Engineering*, vol. 99, 2011.
- [27] P. Bille, "A survey on tree edit distance and related problems," *Theor. Comput. Sci.*, vol. 337, pp. 217-239, 2005.
- [28] A. Nierman and H. V. Jagadish, "Evaluating Structural Similarity in XML Documents," in *Proceedings of the Fifth International Workshop on the Web and Databases (WebDB 2002)*, 2002.
- [29] Y. Chen and Y. Chen, "A new tree inclusion algorithm," *Inf. Process. Lett.*, vol. 98, pp. 253-262, 2006.
- [30] L. L. P. C. M. Weis, "Structure-based inference of xml similarity for fuzzy duplicate detection," presented at the Proceedings of the sixteenth ACM conference on Conference on information and knowledge management, Lisbon, Portugal, 2007.
- [31] H. V. J. Shurug Al-Khalifa , Nick Koudas , Jignesh M. Patel , Divesh Srivastava , Yuqing Wu, "Structural Joins: A Primitive for Efficient XML Query Pattern Matching," presented at the Proceedings of the 18th International Conference on Data Engineering, 2002.
- [32] N. Bruno, N. Koudas, and D. Srivastava, "Holistic twig joins: optimal XML pattern matching," presented at the Proceedings of the 2002 ACM SIGMOD international conference on Management of data, Madison, Wisconsin, 2002.
- [33] H. Jiang, W. Wang, H. Lu, and J. X. Yu, "Holistic twig joins on indexed XML documents," presented at the Proceedings of the 29th international conference on Very large data bases - Volume 29, Berlin, Germany, 2003.
- [34] N. Grimsmo, "Bottom Up and Top Down – Twig Pattern Matching on Indexed Trees," NTNU, 2011.

- [35] N. Grimsmo, T. A. Björklund, and M. L. Hetland, "Fast optimal twig joins," *Proc. VLDB Endow.*, vol. 3, pp. 894-905, 2010.
- [36] N. Grimsmo, "Faster path indexes for search in XML data," presented at the Proceedings of the nineteenth conference on Australasian database - Volume 75, Gold Coast, Australia, 2007.
- [37] P. R. Rao, "Indexing xml data for efficient twig pattern matching," University of Arizona, 2007.
- [38] P. Rao and B. Moon, "Sequencing XML data and query twigs for fast pattern matching," *ACM Trans. Database Syst.*, vol. 31, pp. 299-345, 2006.
- [39] G. Salton and M. J. McGill, *Introduction to Modern Information Retrieval*: McGraw-Hill, Inc., 1986.
- [40] C. Zhang, J. Naughton, D. DeWitt, Q. Luo, and G. Lohman, "On supporting containment queries in relational database management systems," *SIGMOD Rec.*, vol. 30, pp. 425-436, 2001.
- [41] J. Liu, Z. M. Ma, and L. Yan, "Efficient processing of twig pattern matching in fuzzy XML," presented at the Proceedings of the 18th ACM conference on Information and knowledge management, Hong Kong, China, 2009.
- [42] J. Liu, Z. M. Ma, and L. Yan, "FTwig: Efficient algorithm for processing fuzzy XML twig pattern matching," presented at the FSKD, 2010.
- [43] N. Polyzotis, M. Garofalakis, and Y. Ioannidis, "Approximate XML query answers," presented at the Proceedings of the 2004 ACM SIGMOD international conference on Management of data, Paris, France, 2004.
- [44] I. Sanz, M. Mesiti, G. Guerrini, and R. Berlanga, "Fragment-based approximate retrieval in highly heterogeneous XML collections," *Data Knowl. Eng.*, vol. 64, pp. 266-293, 2008.
- [45] I. Sanz, M. Mesiti, G. Guerrini, and R. B. Llavori, "Approximate subtree identification in heterogeneous XML documents collections," presented at the Proceedings of the Third international conference on Database and XML Technologies, Trondheim, Norway, 2005.

- [46] A. Algergawy, R. Nayak, and G. Saake, "Element similarity measures in XML schema matching," *Information Sciences*, vol. 180, pp. 4975-4998, 2010.
- [47] N. Agarwal, M. G. Oliveras, and Y. Chen, "Approximate Structural Matching over Ordered XML Documents," presented at the Proceedings of the 11th International Database Engineering and Applications Symposium, 2007.
- [48] P. Rao and B. Moon, "PRIX: Indexing And Querying XML Using Prefix Sequences," presented at the Proceedings of the 20th International Conference on Data Engineering, 2004.
- [49] H. Wang and X. Meng, "On the Sequencing of Tree Structures for XML Indexing," presented at the Proceedings of the 21st International Conference on Data Engineering, 2005.
- [50] H. Wang, S. Park, W. Fan, and P. S. Yu, "ViST: a dynamic index method for querying XML data by tree structures," presented at the Proceedings of the 2003 ACM SIGMOD international conference on Management of data, San Diego, California, 2003.
- [51] Z. Bao, T. W. Ling, J. Lu, and B. Chen, "SemanticTwig: a semantic approach to optimize XML query processing," presented at the Proceedings of the 13th international conference on Database systems for advanced applications, New Delhi, India, 2008.
- [52] H. Jiang, H. Lu, and W. Wang, "Efficient processing of XML twig queries with OR-predicates," presented at the Proceedings of the 2004 ACM SIGMOD international conference on Management of data, Paris, France, 2004.
- [53] X. Xu, Y. Feng, and F. Wang, "Efficient Processing of XML Twig Queries with All Predicates," presented at the Proceedings of the 2009 Eighth IEEE/ACIS International Conference on Computer and Information Science, 2009.
- [54] J. Lu, T. W. Ling, Z. Bao, and C. Wang, "Extended XML Tree Pattern Matching: Theories and Algorithms," *IEEE Trans. on Knowl. and Data Eng.*, vol. 23, pp. 402-416, 2011.
- [55] Q. Zeng, X. Jiang, and H. Zhuge, "Adding logical operators to tree pattern queries on graph-structured data," *Proc. VLDB Endow.*, vol. 5, pp. 728-739, 2012.



- [56] B. Chen, J. Lu, and T. W. Ling, "Exploiting ID references for effective keyword search in XML documents," presented at the Proceedings of the 13th international conference on Database systems for advanced applications, New Delhi, India, 2008.
- [57] L. Chen, A. Gupta, and M. E. Kurul, "Stack-based algorithms for pattern matching on DAGs," presented at the Proceedings of the 31st international conference on Very large data bases, Trondheim, Norway, 2005.
- [58] H. Wang, J. Li, W. Wang, and X. Lin, "Coding-based Join Algorithms for Structural Queries on Graph-Structured XML Document," *World Wide Web*, vol. 11, pp. 485-510, 2008.
- [59] Z. Vagena, M. M. Moro, and V. J. Tsotras, "Twig query processing over graph-structured XML data," presented at the Proceedings of the 7th International Workshop on the Web and Databases: colocated with ACM SIGMOD/PODS 2004, Paris, France, 2004.
- [60] H. Wu, T. W. Ling, G. Dobbie, Z. Bao, and L. Xu, "Reducing graph matching to tree matching for XML queries with ID references," presented at the Proceedings of the 21st international conference on Database and expert systems applications: Part II, Bilbao, Spain, 2010.
- [61] H. V. Jagadish, L. V. S. Lakshmanan, D. Srivastava, and K. Thompson, "TAX: A Tree Algebra for XML," presented at the Revised Papers from the 8th International Workshop on Database Programming Languages, 2002.
- [62] T. Asai, K. Abe, S. Kawasoe, H. Arimura, H. Sakamoto, and S. Arikawa, "Efficient Substructure Discovery from Large Semi-structured Data," in *SDM*, 2002.
- [63] C. Yu and L. Popa, "Constraint-based XML query rewriting for data integration," presented at the Proceedings of the 2004 ACM SIGMOD international conference on Management of data, Paris, France, 2004.
- [64] A. Bonifati, E. Chang, T. Ho, L. V. Lakshmanan, R. Pottinger, and Y. Chung, "Schema mapping and query translation in heterogeneous P2P XML databases," *The VLDB Journal*, vol. 19, pp. 231-256, 2010.

- [65] X. Yang, M. L. Lee, T. W. Ling, and G. Dobbie, "A semantic approach to query rewriting for integrated XML data," presented at the Proceedings of the 24th international conference on Conceptual Modeling, Klagenfurt, Austria, 2005.
- [66] L. A. Zadeh, "The concept of a linguistic variable and its application to approximate reasoning—I," *Information Sciences*, vol. 8, pp. 199-249, 1975.
- [67] K. T. Atanassov, *Intuitionistic fuzzy sets: theory and applications*: Physica-Verlag, 1999.
- [68] K. Atanassov, "On Intuitionistic fuzzy graphs and Intuitionistic fuzzy relations," *IFSA World Congress*, vol. 1, pp. 551-554, July 1995 1995.
- [69] K. Atanassov, "Temporal intuitionistic fuzzy graphs," *Notes on Intuitionistic Fuzzy Sets*, vol. 4, pp. 59-61, 1998.
- [70] K. T. Atanassov, "Intuitionistic Fuzzy Sets," in *Studies in Fuzziness and Soft Computing*. vol. 35, ed Berlin: Springer, 1999, pp. 121-324.
- [71] K. Atanassov, "On index matrix interpretations of Intuitionistic fuzzy graphs," *Notes on Intuitionistic Fuzzy Sets*, vol. 8, pp. 73-78, 2002.
- [72] M. A. P. Chountas, A. Shannon, K. Atanassov, "On Intuitionistic fuzzy trees," presented at the Notes on Intuitionistic Fuzzy Sets, 2009.
- [73] U. Zernik, *Lexical Acquisition: Exploiting On-line Resources To Build A Lexicon*: Taylor & Francis, 1991.
- [74] P. K. a. P. Prochazkova. (2010, 14/12/2011). *Linguatools*. Available: [http://www.linguatools.de/disco/disco\\_en.html](http://www.linguatools.de/disco/disco_en.html)
- [75] M. Alzebdi, P. Chountas, and K. Atanassov, "Intuitionistic Fuzzy XML Query Matching," in *Flexible Query Answering Systems*. vol. 7022, H. Christiansen, G. Tré, A. Yazici, S. Zadrozny, T. Andreasen, and H. Larsen, Eds., ed: Springer Berlin Heidelberg, 2011, pp. 306-317.
- [76] M. Alzebdi, P. Chountas, and K. T. Atanassov, "An IFTr approach to approximate XML query matching," presented at the SMC, 2011.

- [77] M. Alzebdi, P. Chountas, and K. Atanassov, "Approximate XML Query Matching and Rewriting Using Intuitionistic Fuzzy Trees," presented at the IEEE IS Sofia, 2012.
- [78] R. Bourret. (2009, 12-08-2011). *DTD Parser*. Available: <http://www.rpbourret.com/xmldbms/docs20/org.xmlmiddleware.schemas.dtds.html>

## Appendix A: Publications

The author has published the following publications in relation to this thesis:

### Conference Papers:

- M. Alzebdi, P. Chountas, K. Atanassov, “Approximate XML Query Matching and Rewriting Using Intuitionistic Fuzzy Trees”, IEEE IS 2012. Vol. II, pp. 200-205, 2012.
- M. Alzebdi, P. Chountas, K. Atanassov, “An IFTr Approach to Approximate XML Query Matching”, IEEE SMC 2011, pp. 2425-2430, 2011.
- M. Alzebdi, P. Chountas, K. Atanassov, “Enhancing DWH Models with the Utilisation of Multiple Hierarchical Schemata”, IEEE SMC 2010, pp. 488-492, 2010.
- P. Chountas, M. Alzebdi, A. Shannon, K. Atanassov, “On Intuitionistic Fuzzy Trees”, Notes on Intuitionistic Fuzzy Sets, Vol. 15, No. 2, pp. 30-32, 2009.

### Book Sections:

- M. Alzebdi, P. Chountas, K. Atanassov, “Intuitionistic Fuzzy XML Query Matching”, Springer LNCS 2011. Vol. 7022, pp. 306-317, 2011.